# Elemente de arhitectură a sistemelor de calcul şi operare

# Cuprins


Prefață ..... 7

1. Calculatorul şi sistemul de operare. Noțiuni introductive ..... 9

1.1. Introducere ..... 9

1.2. Noțiuni preliminarii ..... 10

1.3. Principalele componente ale unui sistem de operare ..... 13

1.4. Resursele fizice ale unui sistem de calcul ..... 15

1.4.1. Memoria ..... 15

1.4.2. Microprocesorul ..... 18

1.4.3. Dispozitive de stocare (memoria secundară) ..... 19

1.4.4. Dispozitive de intrare – ieşire ..... 19

1.5. Modalitatea generală de funcționare a unui calculator ..... 21

1.6. Software ..... 22

1.7. Limbaje de nivel înalt şi de nivel jos. Instrucțiuni ..... 23

1.8. Compilatoare şi asambloare ..... 24

1.9. Editarea de legături ..... 25

1.10. Interpretoare ..... 26

1.11. Monoprocesare şi multiprocesare. Definiții ..... 27

1.12. Rezumat ..... 29

2. Microprocesorul ..... 30

2.1. Microprocesorul, componenta principală a calculatorului ..... 30

2.2. Ciclul fetch – decode – execute ..... 33

2.3. Familia de procesoare X86 ..... 47

2.3.1. Microprocesorul 80286 (286) ..... 49

2.3.2. Microprocesorul 80386 (386) ..... 51

2.3.3. Microprocesorul 80486 (486) ..... 53

2.3.4. Microprocesorul Pentium ..... 54

2.3.5. Microprocesorul Pentium MMX ..... 58
2.3.6. Microprocesorul Pentium II („Klamath") ..... 59
2.3.7. Microprocesorul Pentium III ..... 60
2.3.8. Microprocesorul Pentium 4 ..... 60
2.4. Coprocesoare ..... 61
2.4.1. Coprocesoare matematice ..... 61
2.4.2. Tipuri de date admise de către un coprocesor matematic ..... 63
2.4.3. Procesoare DSP (Digital Signal Processor) ..... 65
2.5. Extensii MMX ..... 66
2.6. Elemente de arhitectură internă a procesorului ..... 67
2.6.1. Registrele ..... 67
2.6.2. Unitatea de interfață cu magistrala ..... 68
2.6.3. Unitatea de control ..... 68
2.6.4. Unitatea de execuție a întregilor ..... 69
2.6.5. Unitatea de prelucrare în virgulă mobilă ..... 69
2.6.6. Memoria cache de nivel 1 şi controller-ul de memorie cache ..... 70
2.7. Elemente de arhitectură externă a microprocesorului ..... 71
2.7.1. Procesorul şi magistralele de memorie ..... 71
2.7.2. Magistrala sistem – funcții şi caracteristici ..... 72
2.7.3. Magistralele procesorului şi magistralele de memorie ..... 73
2.7.4. Magistrala de date dintre procesor şi memorie ..... 73
2.7.5. Magistrala de adrese dintre procesor şi memorie ..... 74
2.7.6. Mărimea magistralei de adrese pentru diverse procesoare ..... 75
2.8. Ceasuri de sistem ..... 75
2.9. Noțiuni de multiprocesare ..... 77
2.10. Funcționarea şi operarea întreruperilor ..... 78
2.10.1. Generalități referitoare la întreruperi ..... 78
2.10.2. Controllere de întreruperi ..... 81
2.10.3. Linii IRQ şi magistrala de sistem ..... 82
2.10.4. Întreruperi nemascabile ..... 83

2.11. Procesoarele RISC ..... 84
2.11.1. Introducere ..... 84
2.11.2. Tipuri de procesoare RISC ..... 88
2.11.2.1. Procesoarele din seria MIPS (Silicon Graphics) ..... 88
2.11.2.2. Procesoarele din seria SPARC (Sun Microsystem) ..... 89
2.11.2.3. Procesorul PA-RISC (Hewlett Packard) ..... 91
2.11.2.4. Procesoarele PowerPC (IBM şi Motorola) ..... 91
2.11.2.5. Procesoarele Alpha-DEC (Digital) ..... 92
2.11.3. Concluzii şi viitorul procesoarelor RISC ..... 92
3. Introducere în limbajul de asamblare INTEL ..... 93
3.1. Elemente arhitecturale de bază ale ale microprocesorului INTEL ..... 93
3.1.1. Regiştrii microprocesorului INTEL ..... 93
3.1.1.1. Regiştri de uz general ..... 94
3.1.1.2. Registrul pointer de instrucțiuni (IP) ..... 105
3.1.1.3. Registrul indicatorilor de stare (FLAGS) ..... 107
3.1.1.4. Regiştrii de segment ..... 108
3.2. Elemente ale limbajului de asamblare ..... 111
3.2.1. Formatul general al unei instrucțiuni în limbaj de asamblare ..... 111
3.2.2. Nume de variabile şi etichete ..... 111
3.2.3. Directive de segment simplificate ..... 115
3.2.4. Adrese de memorie şi valori ..... 122
3.2.5. Instrucțiuni ale microprocesorului INTEL ..... 125
3.2.5.1. Instrucțiuni logice ..... 125
3.2.5.2. Instrucțiuni de deplasare şi de rotație ..... 128
3.2.5.3. Instrucțiuni aritmetice ..... 132
3.2.5.4. Instrucțiuni de salt ..... 136
3.3. Exemple de programe ..... 141
4. Memoria şi magistralele de sistem ..... 150
4.1. Introducere ..... 150
4.2. Memoria cache ..... 152

4.3. Regiuni şi partiții de memorie ..... 156
4.3.1. Segmentarea ..... 158
4.3.2. Paginarea ..... 159
4.3.3. Segmentare şi paginare ..... 160
4.4. Modalități de stocare a datelor în memorie ..... 161
4.5. Memoria virtuală ..... 164
4.5.1. Introducere ..... 164
4.5.2. Cererea de pagini ..... 169
4.5.3. Tratarea fenomenului de page-fault ..... 170
4.5.4. Algoritmi de înlocuire a paginilor ..... 173
4.5.4.1. Algoritmul FIFO ..... 173
4.5.4.2. Algoritmul de înlocuire optimă ..... 174
4.5.4.3. Algoritmul LRU (Least Recently Used) ..... 174
4.5.5. Suprapuneri ..... 175
4.5.6. Setarea memoriei virtuale ..... 177
4.6. Magistrale de sistem ..... 178
4.6.1. Arhitectura monomagistrală ..... 179
4.6.2. Arhitectura multimagistrală ..... 182
4.7. Memoria externă ..... 183
4.7.1. Hard-diskul ..... 183
4.7.2. Discuri optice ..... 186
4.7.2.1. CD-ROM (Compact Disk-Read Only Memory) ..... 186
4.7.2.2. DVD (Digital Versatile Disk – Digital Video Disk) ..... 191
4.7.3. Discuri floppy ..... 193
5. Rețele de calculatoare ..... 195
5.1. Introducere ..... 195
5.2. Topologii ..... 195
5.2.1. Topologii fizice ..... 196
5.2.2. Topologii logice ..... 198
5.2.2.1. Topologia broadcast ..... 198

5.2.2.2. Topologia token passing ..... 199
5.3. Echipamente de rețea ..... 200
5.4. Modele utilizate în dezvoltarea şi studiul rețelelor de calculatoare ..... 205
5.4.1. Introducere ..... 205
5.4.2. Necesitatea unui model ierarhic ..... 206
5.4.3. Modelul ISO – OSI ..... 207
5.4.3.1. Nivelul aplicație ..... 208
5.4.3.2. Nivelul prezentare ..... 209
5.4.3.3. Nivelul sesiune ..... 210
5.4.3.4. Nivelul transport ..... 211
5.4.3.5. Nivelul rețea ..... 216
5.4.3.6. Nivelul legăturii de date ..... 218
5.4.3.7. Nivelul fizic ..... 219
5.4.4. Modelul TCP – IP ..... 220
5.4.4.1. Introducere ..... 220
5.4.4.2. Nivelul aplicație ..... 222
5.4.4.3. Nivelul transport ..... 222
5.4.4.4. Nivelul Internet ..... 222
5.4.4.5. Nivelul gazdă – la – rețea ..... 223
5.5. Internetul şi rețelele de calculatoare ..... 223
5.5.1. Introducere şi concepte de bază ..... 223
5.5.2. Internet – scurt istoric ..... 226
5.5.3. Definiția unui protocol de rețea ..... 227
5.5.4. Naşterea stivei de protocoale TCP-IP ..... 229
5.5.5. Arhitectura Internetului ..... 230
5.5.6. Componente Internet ..... 232
5.5.6.1. Calculatoare gazdă, clienți şi servere ..... 232
5.5.6.2. Servicii orientate pe conexiune ..... 234
5.5.6.3. Servicii neorientate pe conexiune ..... 235
Bibliografie ..... 236

# *Prefață*

Această carte prezintă o serie dintre elementele organizatorice şi arhitecturale ale calculatoarelor, presupunându-se că cei care o folosesc au deja noțiuni preliminarii legate de bazele aritmeticii, bazele logicii booleene şi circuite digitale. În acest sens, cartea este concepută pentru a fi utilizată în cadrul cursurilor de organizarea şi arhitectura calculatoarelor.

În capitolul 1 sunt prezentate noțiuni introductive legate de calculator şi sistemul de operare, memorie, microprocesor, dispozitive de stocare, dispozitive de intrare/ieşire, modalitatea generală de funcționare, limbaje de nivel înalt şi nivel jos, asambloare şi interpretoare.

Capitolul al doilea abordează microprocesorul, componenta principală a calculatorului. La începutul capitolului sunt prezentate ciclul de bază al funcționării microprocesorului şi familia de procesoare x86, continuându-se cu prezentarea coprocesoarelor şi a elementelor de arhitectură internă şi externă a procesorului. În finalul capitolului sunt oferite noțiuni de multiprocesare, funcționarea şi operarea întreruperilor şi arhitectura procesoarelor RISC.

În capitolul 3 este făcută o introducere în limbajul de asamblare Intel, prezentându-se aici elementele arhitecturale de bază ale microprocesorului Intel, formatul unei instrucțiuni în limbaj de asamblare, directive de segmentare simplificate, precum şi o serie dintre cele mai utilizate instrucțiuni ale microprocesorului Intel împreună cu nişte exemple de programe.

Capitolul al 4-lea continuă cu prezentarea memoriei unui sistem de calcul, a memoriei cache şi a modalităților de utilizare eficientă a memoriei prin intermediul paginării şi segmentării. Sunt prezentate, de asemenea, modalități de stocare a datelor în memorie,

noțiuni legate de administrarea memoriei virtuale, magistralele de sistem şi memoria externă.

Datorită faptului că în zilele noastre practic nu mai există calculatoare de sine stătătoare ci calculatoare conectate la rețea (fie la Internet, fie la rețeaua locală LAN), în ultimul capitol sunt prezentate noțiuni de bază referitoare la rețelele de calculatoare. La începutul capitolului sunt înfățișate topologiile de rețea, apoi se continuă cu prezentarea echipamentelor de rețea şi a modelelor utilizate în dezvoltarea şi studierea rețelelor de calculatoare. În finalul capitolului sunt prezentate arhitectura Internetului şi componentele acestuia.

Recomand această carte în primul rând studenților din anul I de la Facultatea de Cibernetică, Statistică şi Informatică Economică din cadrul ASE Bucureşti ce urmează cursul de *Sisteme de Calcul şi Operare*, precum şi tuturor acelor studenți ce studiază cursuri de arhitectură a calculatoarelor şi a sistemelor de operare.

Autorul

# 1

# CALCULATORUL ŞI SISTEMUL DE OPERARE. NOȚIUNI INTRODUCTIVE

## 1.1 Introducere

În ultimii ani calculatoarele au devenit, indiscutabil, componente vitale ale societății, fiind prezente în activități din cele mai diverse în industrie, economie, educație, sănătate, cercetare; practic în orice domeniu economic sau social al societății. De asemenea, calculatoarele au produs o nouă revoluție pentru civilizație, revoluția informațională adusă de acestea încadrându-se după revoluțiile din agricultură şi din industrie. Contribuind la creşterea puterii intelectuale a omenirii, calculatoarele au afectat şi afectează în continuare toate domeniile investigației ştiințifice, cercetarea computațională conlucrând cu cea teoretică şi cea experimentală în explorarea de noi frontiere ale cunoaşterii în cele mai diverse domenii: biologie, chimie, astronomie, medicină, etc. Revoluția din domeniul calculatoarelor evoluează continuu; aplicații care până ieri erau de domeniul ştiințifico-fantasticului sunt astăzi aplicații banale, de la automatele bancare la microprocesoarele integrate în automobile, de la calculatoarele mobile ultra-compacte şi miniaturizate la Internet şi World Wide Web.

Cartea de față îşi propune să înfățişeze aspectele mai ascunse, de detaliu ale acestei "maşinării" extraordinare care este calculatorul personal. Sunt prezentate aici elementele arhitecturale vizibile programatorului (mulțimea de instrucțiuni a procesorului, numărul de biți utilizați pentru reprezentarea datelor, mecanisme de intrare/ieşire, tehnici de adresare, etc.) precum şi câteva elemente organizatorice (funcționalități legate de semnale de control, interfețe, tehnologii de memorie). Primul capitol tratează aspectele fundamentale, ideile şi definițiile de bază legate de componentele hardware şi software ale calculatoarelor.

## 1.2 Noţiuni preliminarii

Calculatorul este un dispozitiv extrem de complex; pentru a putea înţelege mai bine arhitectura sa şi modalitatea de funcţionare, de regulă se apelează la o împărţire ierarhică pe componente. Aceste componente ierarhice pleacă de la nivelul cel mai de jos (nivelul cel mai apropiat de structura hardware-fizică a calculatorului) şi continuă apropierea de utilizator prin considerarea elementelor nivelului mai înalt (nivelul apropiat de componenta software-programe a calculatorului). Componentele unui astfel de model ierarhic sunt prezentate în figura 1.1.

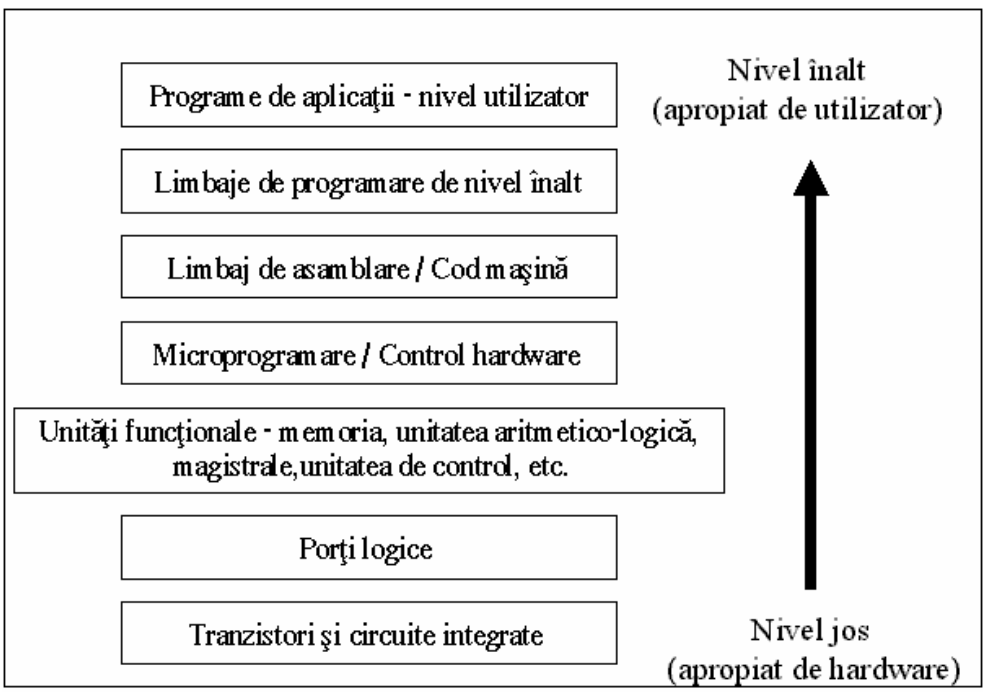

Programe de aplicaţii - nivel utilizator
Limbaje de programare de nivel înalt
Limbaj de asamblare / Cod maşină
Microprogramare / Control hardware
Unităţi funcţionale - memoria, unitatea aritmetico-logică, magistrale, unitatea de control, etc.
Porţi logice
Tranzistori şi circuite integrate
Nivel înalt
(apropiat de utilizator)
Nivel jos
(apropiat de hardware)

Fig. 1.1 Niveluri ierarhice în studiul calculatoarelor

Aşa cum se observă în acest desen, nivelurile inferioare sunt constituite de componentele hardware (tranzistori, circuite integrate, porţi logice) ce stau la baza construcţiei calculatorului. Urmează apoi unităţile funcţionale ale microprocesorului (unitatea de control ce administrează întreaga funcţionare, unitatea aritmetico-logică ce execută operaţiile aritmetice şi logice, memoria, magistralele de conexiune etc.) şi ne apropiem de nivelul superior al programelor de aplicaţii (cel mai apropiat de utilizator) prin nivelul microprogramat, nivelul limbajului de asamblare şi nivelul limbajelor de programare de nivel înalt.

Din punct de vedere fizic, un calculator este constituit din patru componente de bază:

- *microprocesorul*, denumit şi *UCP (Unitatea Centrală de Procesare)*, este componenta care controlează modalitatea de funcţionare a calculatorului şi execută şi operaţiile de procesare a datelor;
- *memoria principală* – necesară pentru stocarea datelor;
- *interfeţele de intrare/ieşire* – necesare pentru asigurarea interacţiunii calculatorului cu mediul extern (introducere de date/oferirea rezultatelor);
- *componenta de conexiune* între componente – *magistrala de sistem* este mecanismul ce oferă modalitatea de comunicare între componentele sistemului.

Aceste componente sunt ilustrate în figura 1.2.

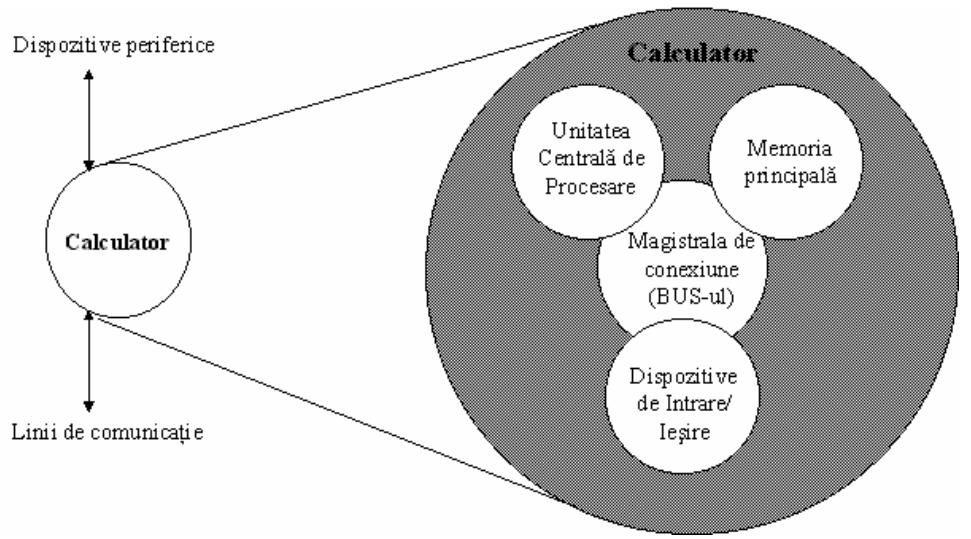



Fig. 1.2 Componentele arhitecturale de bază ale unui calculator

Pentru a putea funcţiona, un computer are nevoie, pe lângă componentele fizice din care este alcătuit, de nişte programe special scrise pentru a putea rezolva în mod automatizat anumite sarcini. Fără existenţa unor programe speciale un calculator nu poate fi utilizat, nu se pot introduce date de la tastatură, nu se pot afişa rezultate pe ecranul monitorului şi nu se poate porni un program de aplicaţie sau, dacă vreţi, un joc pe calculator. Totalitatea resurselor fizice de care dispune calculatorul formează

componenta *hardware*, pe când celelalte resurse logice, formate din diverse programe şi rutine formează componenta *software*.

Componenta software cuprinde două categorii distincte de programe: *programele de aplicaţii* şi *programele de sistem*, sau, pe scurt, *sistemul de operare*. Sistemul de operare este constituit dintr-o serie de rutine software ce asigură interfaţa între componenta hardware şi programele de aplicaţii (figura 1.3). Toate celelalte componente software sunt guvernate de către sistemul de operare şi rulează sub "îndrumarea" acestuia. Accesul la componentele hardware ale sistemului (calculatorului) se face numai prin intermediul sistemului de operare. Sistemul de operare are, deci, rolul de a asigura interfaţa între hardware şi software; din această cauză, programarea aplicaţiilor se face ţinând cont de acest rol primordial al sistemului de operare. Programatorii de software de aplicaţii pot astfel utiliza diverse rutine ale sistemului de operare pentru a avea un acces mai simplu la operaţiile de intrare/ieşire cu partea hardware a sistemului.

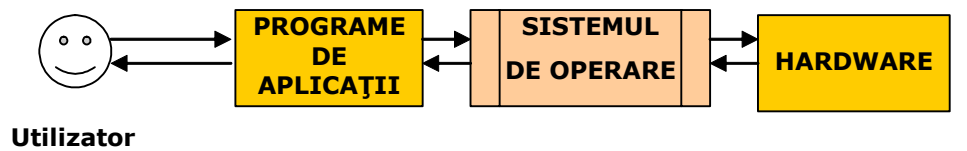



Fig. 1.3 Sistemul de operare asigură interfaţa între hardware şi programele de aplicaţii

Componentele unui sistem de operare oferă funcţionalităţi diverse, cum ar fi asigurarea comunicării cu dispozitivele periferice (operaţiile de intrare/ieşire) sau preluarea de comenzi de la utilizator şi execuţia diverselor comenzi (crearea unui fişier sau director, lansarea în execuţie a unui program, accesarea memoriei, adăugarea unei noi componente hardware calculatorului, etc.). Majoritatea acestor sarcini reprezintă cerinţe "sine qua non" pentru toate programele de aplicaţie.

Deoarece programele de aplicaţie accesează partea hardware prin intermediul sistemului de operare, rolul acestuia din urmă este, prin urmare, acela al unui depozit sau biblioteci ce conţine astfel de rutine ce asigură accesul la dispozitivele hardware ale calculatorului. Existenţa unei astfel de biblioteci de rutine software pentru acces la componenta hardware asigură simplificarea programării la nivelul software-ului de aplicaţie, având în vedere că sarcinile legate de operaţiile de intrare/ieşire nu sunt totdeauna dintre cele mai simplu de programat.

Din punct de vedere hardware, computerele provenite de la diverşi producători nu sunt compatibile între ele 100%. De aceea, un program

software scris pentru un anumit calculator nu va rula pe un calculator provenit de la un producător ce utilizează un alt tip de microprocesor (denumit şi UCP – Unitatea Centrală de Procesare), spre exemplu. Este cazul şi sistemelor de operare: un sistem de operare scris pentru un calculator de tipul IBM-PC nu va rula pe un calculator SUN-Sparc şi vice-versa. Deoarece componentele hardware sunt diferite iar sistemul de operare reprezintă interfața între software şi hardware, şi interfața trebuie să fie diferită, drept urmare sistemul de operare trebuie să conțină rutine diferite. Aceste rutine ale sistemului de operare ce comunică direct cu partea hardware pot fi diferite însă ele constituie o platformă de interfață consistentă pentru asigurarea comunicării cu programele de aplicații. Datorită acestei interfețe, programatorii de aplicații pot ignora diferențele hardware şi se pot concentra asupra aplicației în sine.

## 1.3 Principalele componente ale unui sistem de operare

Evoluția din punct de vedere hardware a computerelor a determinat şi o evoluție la nivel software. Din acest punct de vedere, sistemele de operare moderne au o serie de componente principale (figura 1.4), asigurând următoarele funcții de bază:

- Interfața cu utilizatorul;
- Managementul memoriei;
- Managementul fişierelor;
- Managementul microprocesorului;
- Managementul dispozitivelor periferice.

Acest capitol fiind unul introductiv, vom prezenta în continuare în linii mari caracteristicile generale ale acestor cinci funcțiuni de bază oferite de către un sistem de operare modern.

***Interfața cu utilizatorul.*** Componenta sistemului de operare ce asigură interfața cu utilizatorul (denumită în marea majoritate a cazurilor *componenta shell*), oferă modalitățile prin care utilizatorul şi programele de aplicații pot comunica cu sistemul de operare şi pot efectua cereri de servicii către acesta. Interfețele simple „la linia de comandă” din MS-DOS şi UNIX sunt exemplele clasice, în timp ce în ultimii ani s-au impus interfețele grafice de tip GUI (Graphical User Interface) ale sistemelor de operare Windows, Macinstosh sau interfețele (de asemenea grafice) de tip X-Window (KDE, Gnome, etc.) ale diverselor variante de Linux sau UNIX. Prin intermediul interfețelor grafice se pot selecta simboluri grafice (icon-uri) cu ajutorul mouse-ului, se pot selecta opțiuni din meniurile ferestrelor grafice, se pot lansa în execuție programe etc.

***Managementul fişierelor*** reprezintă capacitatea unui sistem de operare (întâlnită de regulă sub numele de *sistem de fişiere*) de a permite utilizatorilor şi programelor de aplicaţie de a manipula (crea, modifica, redenumi, modifica, şterge, etc.) fişiere şi directoare.

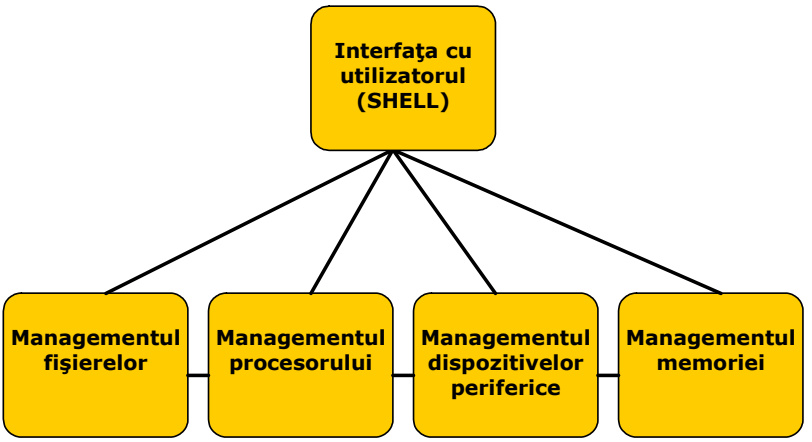



Fig. 1.4 Principalele funcţiuni ale unui sistem de operare

***Managementul microprocesorului*** asigură o bună administrare a modalităţii de utilizare a microprocesorului şi a timpului alocat de acesta diverselor programe în execuţie. Noţiunea de bază când este vorba de timpul alocat de procesor diverselor instanţe ale programelor aflate în execuţie (cunoscute sub denumirea de *procese*) este cea de *ciclu de ceas*.

***Managementul dispozitivelor periferice*** se referă la responsabilitatea sistemului de operare de a administra comunicaţiile cu dispozitivele periferice ale calculatorului, de exemplu cu: tastatura, mouse-ul, ecranul, imprimanta, scanner-ul, modem-ul, unitatea de CD-ROM, etc.

În fine, ultima funcţie de bază, ***managementul memoriei*** este legată de administrarea resurselor de memorie principală a sistemului. Se cunoaşte faptul că pentru a putea fi lansat în execuţie, un program are nevoie de memorie. Prin funcţia de management al memoriei, sistemul de operare asigură în timpul funcţionării calculatorului alocarea spaţiului necesar aplicaţiilor să ruleze, asigurând ca spaţiile de memorie alocate aplicaţiilor să nu interferează între ele sau să nu se suprapună peste spaţiul de memorie alocat însuşi sistemului de operare.

Orice sistem de operare modern, fie că este vorba despre UNIX, Linux sau Windows, oferă toate aceste funcţionalităţi. Chiar dacă acestea au fost prezentate separat, nu înseamnă că ele nu interacţionează între ele; din

contră, toate aceste componente ale unui sistem de operare conlucrează împreună pentru buna funcționare a sistemului de calcul. Spre exemplu, să considerăm cazul în care dorim să deschidem un fişier pentru a-l tipări la imprimantă. Ce operații rezolvă sistemul de operare pentru acest lucru? În primul rând, sistemul de fişiere preia numele fişierului şi determină locația precisă a acestuia în cadrul sistemului de fişiere de pe hard-disk. În continuare, rutinele de management al dispozitivelor periferice folosesc această locație pentru a accesa hard-disk-ul şi a citi fişierul. În acest timp, managementul memoriei se ocupă cu asigurarea spațiului de memorie necesar pentru a stoca temporar fişierul, managementul procesorului oferă suport pentru a ghida în continuare procesorul pentru a iniția rutina de management a dispozitivelor periferice pentru a accesa imprimanta şi a face în final tipărirea. Toată această succesiune complicată de operații se petrece de fiecare dată când deschidem un fişier pentru editare sau pentru tipărire, ne pregătim să trimitem un e-mail sau un fax, totul întâmplându-se transparent față de utilizator.

## 1.4 Resursele fizice ale unui sistem de calcul

### *1.4.1 Memoria*

Memoria unui sistem de calcul se poate împărți în două clase mari de memorie: memoria principală sau internă (pe scurt, memorie) şi memoria secundară sau externă (reprezentată de diferite medii de stocare externe, cum ar fi dischetele, hard-disk-ul, CD-ROM-ul, etc.). Pentru a putea fi executat, un program trebuie mai întâi încărcat în memoria principală. Un sistem de calcul nu poate executa un program direct de pe un suport de memorie extern (din memoria externă) şi nici nu poate manipula date stocate în memoria externă decât dacă acestea au fost încărcate în prealabil în memoria internă.

Memoria principală conține întotdeauna programul curent ce se execută şi datele curente ce sunt procesate. În timp ce memoria internă este o memorie *volatilă*, al cărui conținut se pierde o dată cu închiderea calculatorului, memoria externă reprezintă modalitatea de stocare a datelor pe termen lung, fiind o memorie *nevolatilă*. Memoria internă este de regulă identificată cu aşa-numita memorie *RAM (Random Access Memory),* o memorie care poate fi atât *citită* cât şi *scrisă*.

Operația prin care o valoare este adusă (citită) de la o anumită adresă din memorie poartă numele de *citire* din memorie, pe când operația de stocare a unei valori la o adresă de memorie poartă numele de *scriere* în

memorie. Operația de citire este o operație *nedistructivă*, în sensul în care valoarea citită rămâne stocată în memorie, pe când operația de scriere este o operație *distructivă*, în sensul că valoarea anterioară din memorie se pierde, fiind înlocuită cu noua valoare scrisă în memorie. Conținutul memoriei RAM poate fi atât citit cât şi scris, deci modificat cu uşurință. Programele utilizează memoria RAM pentru a fi executate şi a procesa date.

Un alt tip de memorie este memoria *ROM (Read Only Memory)* care poate fi citită, nu şi scrisă. Acest tip de memorie este folosit pentru a stoca date legate de configurația hardware a calculatorului, aşa-numitul *BIOS (Basic Input Output System),* cipul responsabil de stocarea programului *bootstrap*, ce reprezintă programul inițial de configurare a calculatorului.

O altă categorie de memorie internă este memoria *cache* (pronunțată ca în englezescul „cash"). Pe scurt, memoria cache este o memorie intermediară utilizată pentru stocarea valorilor din memoria RAM înainte de a fi prelucrate de microprocesor. Memoria cache este o memorie foarte rapidă şi este utilizată pentru a mări viteza de procesare a datelor în drumul lor din memoria RAM spre microprocesor.

În principal, mecanismul de funcționare a memoriei cache este următorul: un program în execuția este stocat în memoria RAM, iar instrucțiunile şi datele programului ce vor fi executate într-un timp foarte apropiat sunt aduse întâi în memoria ultrarapidă de tip cache. De aici, datele şi instrucțiunile sunt transferate către microprocesor; în acest mod, microprocesorul va utiliza direct memoria cache care este mult mai rapidă decât memoria RAM şi, drept urmare, viteza de procesare creşte.

Singurul tip de memorie mai rapidă decât memoria cache este memoria internă a procesorului, formată din regiştrii microprocesorului. Mai multe detalii despre arhitectura internă a microprocesorului vom analiza în capitolul consacrat microprocesorului. Prezentăm în figura 1.5 o ierarhie sub formă piramidală a memoriei, în care baza „piramidei" este alcătuită din memoria secundară de tip bandă magnetică iar vârful este format de regiştrii microprocesorului, cu cea mai scăzută capacitate de stocare dar cel mai rapid tip de memorie şi, totodată, cel mai mare preț. În partea din dreapta a figurii sunt prezentate dimensiunile standard ale celor şase tipuri de memorie din ierarhie. În general există două reguli valabile legate de ierarhia de memorie a unui sistem de calcul:

- Memoria rapidă este scumpă;
- Memoria ieftină este mai puțin rapidă;

În consecință, se încearcă minimizarea raportului preț/ performanță având în vedere anumite costuri.

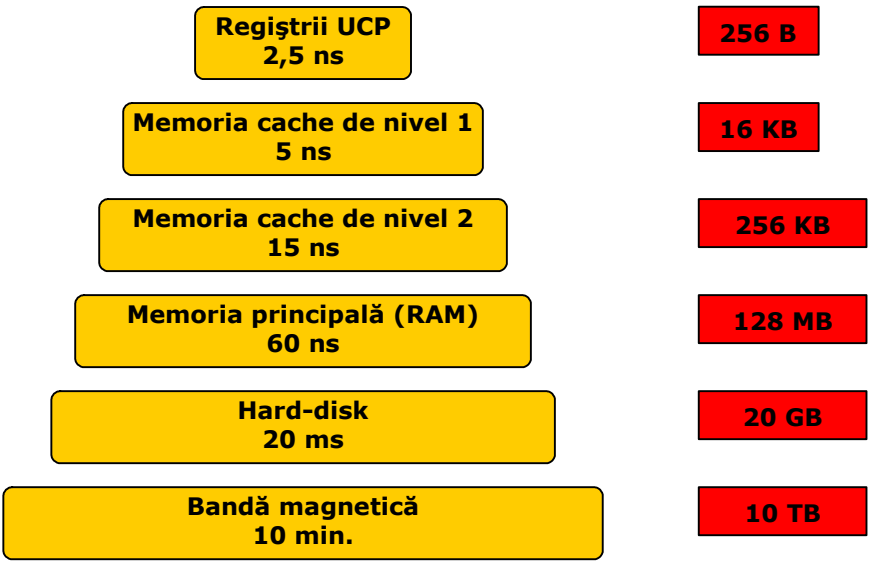



Fig. 1.5 O ierarhie a memoriei

Fiecare tip de memorie din ierarhie este administrat de către o unitate de control: regiştrii microprocesorului sunt administrați de către aşa-numita „unitate de control a UCP", memoria cache de nivel 1 (cache L1) este administrată de controller-ul primar de memorie cache iar memoria cache de nivel 2 (cache L2) este guvernată de către controller-ul secundar de memorie cache. Memoria principală RAM este administrată de către unitatea de management a memoriei (MMU – Memory Management Unit) iar hard-disk-ul şi banda magnetică de către utilizator. De asemenea, la fiecare din cele şase nivele din ierarhie datele sunt grupate în unități de date diferite din punct de vedere al mărimii (ca număr de *octeți* - *bytes*), astfel:

- La nivelul regiştrilor UCP unitatea de date se numeşte *cuvânt (word)* şi poate fi reprezentat pe 2,4 sau 8 octeți;
- La nivelul memoriei cache L1 şi L2 datele sunt structurate în *linii de memorie* 32 de octeți;
- La nivelul memoriei RAM avem de-a face cu *pagini de memorie* de 4 KB;
- La nivelul hard-disk-ului şi al unității de bandă *fişierele* se măsoară de regulă în MB.

**Adresarea memoriei**

Capacitatea de stocare a memoriei este reprezentată în mod uzual prin MB (Megabytes sau Megaocteți), unde 1 MB=1.024 KB=1.048.576 B.

***Observație:*** De regulă, capacitatea de stocare a memoriei se măsoară în multipli de bytes (notați cu litera mare **B**), pe când în comunicații, viteza de transfer a datelor (într-o rețea de calculatoare, spre exemplu), denumită şi *lățime de bandă*, se măsoară în multipli de biți (notați cu litera mică **b**). Astfel, notația **1 KB** reprezintă un **kilobyte**, pe când notația **1 Kb** reprezintă un **kilobit**.

Octeții de memorie (sau grupurile de octeți) au atribuite o serie de adrese de memorie pentru a putea fi accesați. De regulă, octeții sau cuvintele (grupări de octeți) sunt numerotați secvențial: 0,1,2,… În acest mod, procesorul poate avea acces la o anumită zonă de memorie specificând adresa de memorie a acelei zone. Dacă, spre exemplu, microprocesorul trebuie să utilizeze datele stocate în memorie la adresa 1000, nu are de făcut decât să acceseze memoria la adresa 1000. În funcție de tipul de memorie, octetul sau gruparea de octeți (reuniți sub denumirea de *cuvânt - word*) reprezintă unitatea fundamentală adresabilă de memorie. Astfel, datele prelucrate de către microprocesor la un moment dat pot fi sub formă de octeți (bytes) sau sub formă de cuvinte (words).

### *1.4.2 Microprocesorul*

Microprocesorul reprezintă componenta principală a unui calculator, fiind considerat, pe bună dreptate, „creierul" calculatorului. Întâlnit şi sub denumirea de UCP (Unitatea Centrală de Procesare), microprocesorul are nevoie totuşi de un program pentru a putea controla şi procesa datele, de aceea „coeficientul de inteligență" al său depinde într-o mare măsură şi de software. Datele stocate în memorie sunt prelucrate de către microprocesor prin intermediul unui program stocat în memorie. În esență, un *program* reprezintă o *secvență de instrucțiuni* (de genul: adunare, scădere, înmulțire, comparare, etc.) care guvernează modalitatea de prelucrare a datelor de către microprocesor. Fiecare instrucțiune are un cod de operației şi poate avea unul sau mai mulți operanzi, sau nici unul. Codul operației specifică operația ce va fi executată, în timp ce operanzii identifică datele din memorie asupra căror se va aplica acea operație (fie că este vorba despre o adunare, scădere, înmulțire, comparare, etc.). Mai multe detalii despre componentele şi funcționarea microprocesorului vom prezenta în capitolul rezervat acestuia.

### *1.4.3 Dispozitive de stocare (memoria secundară)*

Datorită faptului că memoria RAM este foarte scumpă şi este, după cum am mai spus, o memorie volatilă, pentru stocarea pentru mai mult timp (nu numai pe parcursul funcţionării calculatorului) a datelor s-a căutat utilizarea unui alt tip de memorie. Acest tip de memorie este memoria externă (sau memoria secundară), care este o memorie relativ rapidă, fiabilă, ieftină, de mare capacitate şi, pe deasupra, ne-volatilă. Am văzut mai înainte că un calculator nu poate executa un program decât dacă acesta este încărcat în memoria principală, din acest punct de vedere, memoria principală reprezintă memoria utilizată pentru stocarea programului ce se execută şi a datelor ce sunt prelucrate iar memoria secundară reprezintă memoria folosită pentru stocarea pe termen lung a datelor.

### *1.4.4 Dispozitive de intrare/ieşire*

Dispozitivele de intrare/ieşire (sau dispozitivele periferice) asigură modalitatea prin care utilizatorul are acces la calculator. Dispozitivul de intrare cel mai comun întâlnit este, desigur, tastatura. Pe măsură ce caracterele sunt tastate cu ajutorul tastelor de pe tastatură, acestea sunt stocate în memorie şi apoi copiate din memorie către dispozitivul de bază de ieşire, care este monitorul. În linii mari, ecranul monitorului reprezintă fereastra prin care utilizatorul poate vizualiza respectivul conţinut al memoriei. Imaginea care apare pe ecranul monitorului este, evident, o imagine temporară; în momentul întreruperii alimentării cu curent a monitorului această imagine dispare; putem însă direcţiona această imagine către o imprimantă pentru a obţine o imagine permanentă. Imprimantele cu jet de cerneală sau imprimantele laser reprezintă principalele tipuri de imprimante folosite în practică în prezent.

Alte tipuri de dispozitive periferice de intrare/ieşire sunt reprezentate de dispozitivele magnetice sau de dispozitivele optice. Un exemplu de dispozitiv magnetic îl reprezintă cardurile cu bandă magnetică emise de bănci. Banda magnetică a cardului stochează informaţii ca: numărul de cont al titularului, limita de debit/credit şi data de expirare a cardului, acest tip de card fiind citit într-o manieră asemănătoare cu aceea a citirii unei benzi magnetice audio. Printre dispozitivele periferice optice întâlnite în mod frecvent sunt scannerele de uz general, prin intermediul cărora imagini de pe hârtie sunt preluate şi convertite în format digital ce poate fi prelucrat cu ajutorul calculatorului. Un tip special de scannere îl reprezintă scannerele

(cititoarele) pentru coduri de bare ataşate la casele de marcat din majoritatea supermarket-urilor.

**Dispozitive de interfață**

Datele sunt stocate în calculator sub forma unor şiruri de biți; în acest sens sunt utilizate diverse codificări. La nivelul computerului, aceste reprezentări sunt unice; spre exemplu, dacă în reprezentarea internă a calculatorului caracterul Z are forma: **01011010**, atunci acest şir de biți este folosit în mod unic pentru a reprezenta caracterul Z.

Această regulă nu este însă valabilă pentru dispozitivele periferice sau pentru memoria secundară. În cazul unei tastaturi, fiecare tastă generează un caracter. În cazul unei imprimante, caracterele sunt reprezentate sub formă unor matrice de puncte. Un dispozitiv optic citeşte informația prin intermediul intensității luminoase, în timp ce un dispozitiv magnetic înregistrează şi citeşte porțiuni magnetizate. Fiecare echipament periferic foloseşte o modalitate proprie de reprezentare a datelor iar această reprezentare poate să coincidă sau nu cu reprezentarea internă a datelor în calculator. Dacă aceste reprezentări sunt diferite este necesar un dispozitiv de translatare dintr-o reprezentare în alta. Aici intervine rolul dispozitivului de interfață, numit şi *placa de interfață*.

Considerând cazul tastaturii, atunci când o tastă este apăsată, se transmite un semnal electronic către interfața tastaturii. Ca răspuns la semnalul electronic primit, interfața tastaturii generează codul ce reprezintă caracterul stocat în interiorul calculatorului, transferând acest cod în memoria calculatorului. În cazul unei imprimante, şirurile de biți transmise dinspre calculator spre imprimantă sunt preluate de interfața imprimantei, care translatează aceste şiruri de biți în reprezentarea recunoscută de către imprimantă pentru a se putea face tipărirea. Cele două dispozitive periferice considerate aici, tastatura şi imprimanta, reprezintă două dispozitive diferite din punct de vedere al reprezentării datelor; datorită existenței dispozitivelor de interfață, ele pot fi conectate la acelaşi calculator. La intrare, interfața translatează semnalele externe într-un format ce este recunoscut de către calculator. În cazul semnalelor de ieşire, acestea sunt convertite din formatul intern al calculatorului în formatul propriu al dispozitivului de ieşire respectiv.

Şi în cazul dispozitivelor de memorie secundară legătura dintre calculator şi acestea se face prin intermediul interfețelor. Interfața controlează din punct de vedere fizic dispozitivul de disc, acceptând comenzi de citire, scriere, căutare din partea microprocesorului. Atribuirea unei interfețe fiecărui echipament conectat la un calculator este un lucru practic în cazul calculatoarelor personale.

Pentru calculatoare ce pot avea sute sau mii de echipamente periferice (cazul computerelor *mainframe*) conectate, modalitatea de comunicare dintre calculator şi dispozitivele periferice se schimbă: sunt utilizate canale şi unităţi de control I/O specifice. În acest caz, fiecare dispozitiv fizic are propria unitate de control iar canalul asigură comunicaţia cu calculatorul, în timp ce unitatea de control comunică cu echipamentul extern în limbajul acestuia din urmă. Canalul şi unităţile de control acţionează împreună pentru a realiza operaţia de translatare în ambele sensuri.

## 1.5 Modalitatea generală de funcţionare a unui calculator

Pentru a putea analiza atât structura cât şi modalitatea de funcţionare ale unui computer digital, trebuie să vedem care sunt etapele necesare unui program pentru a ajunge să fie executat în memoria calculatorului. Vom discuta în continuare despre *asambloare, compilatoare, editoare de legături şi interpretoare*.

Reamintim câteva date importante din istoria computerelor:

**Blaise Pascal – 1642** – inventează primul calculator mecanic;

**Charles Babbage – 1830** – a construit două maşini (funcţionau cu aburi!) de calcul: "Difference Engine" şi "Analytical Engine";

Fundamentele structurale ale generaţiei actuale de computere digitale se bazează pe arhitectura elaborată de **John von Neumann** în anii **1940** (Figura 1.6).

Fig. 1.6 Modelul procesării datelor al lui John von Neumann

Ideea originală a lui von Neumann a fost aceea a unui program executabil care să controleze activitatea unei maşini de uz-general. În acest caz programul este o listă de instrucţiuni folosite într-un anumit scop iar computerele folosesc memoria internă pentru a stoca atât programe cât şi date.

Vom vedea că pentru a face distincţie între programe şi date (în final acestea se rezumă la şiruri de biţi) trebuie ca acestea să fie stocate în memorie în regiuni specifice diferite.

Activitatea complexă a unui computer este posibilă prin strânsă interacţiune între hardware şi software. Software-ul este stocat în memorie iar unitatea centrală de procesare (UCP - procesorul) reprezintă partea hardware ce execută practic instrucţiunile unui program. Este interesant de remarcat faptul că ideea de bază a lui von Neumann a rămas neschimbată de peste 50 de ani: un computer conţine un program modificabil aflat în memoria unificată şi guvernează activităţile sale operaţionale.

O mică modificare a fost totuşi adusă acestui concept de bază: aşa numita **"ARHITECTURĂ HARVARD"**, care separă datele de programe şi necesită memorii distincte pentru acestea, cât şi magistrale de acces diferite. Scopul acestei separări este acela de a creşte ratele de transfer şi de a îmbunătăţi (creşte) cantitatea de date procesate în unitatea de timp.

## 1.6 Software

Un *program* reprezintă o serie de *instrucţiuni* ce coordonează activitatea unui calculator într-un şir de etape. Fiecare instrucţiune "spune" calculatorului să execute una dintre operaţiile de bază: adunare, scădere, înmulţire, împărţire, comparaţie, citire, scriere sau copiere. Pentru a realiza acest lucru, procesorul aduce o instrucţiune din memorie pe care o execută în timpul unui ciclu maşină. De regulă, o instrucţiune conţine un *cod al operaţiei* ce specifică funcţia pe care trebuie să o îndeplinească şi o serie de operanzi care identifică locaţiile de memorie sau regiştrii de memorie ce conţin datele ce vor fi manipulate. Fiind dată instrucţiunea: **ADD 5,6**, aceasta poate spune unui calculator să adune conţinutul regiştrilor 5 şi 6. În acest caz, codul operaţiei este ADD.

De regulă, mulţimea de instrucţiuni pe care le înţelege un procesor este limitată. Pentru a aduna două numere, de fapt sunt utilizate în mod normal nu mai puţin de patru instrucţiuni, deoarece valorile numerelor trebuie aduse din memorie şi încărcate în regiştri, apoi valorile din regiştri sunt adunate şi răspunsul este stocat. În acest caz, cele patru instrucţiuni sunt: LOAD, LOAD, ADD şi STORE.

Un calculator rulează sub controlul unui program stocat în memorie sub formă de biţi. Din această cauză, pentru a fi executat, programul trebuie să existe sub formă de biţi (formă binară). La începuturile programării primelor calculatoare, programatorii scriau programele chiar sub forma unor şiruri de biţi; lucru deosebit de dificil. Apariţia limbajelor de programare, a asambloarelor, compilatoarelor şi interpretoarelor a schimbat, din fericire, acest mod de lucru al programatorilor.

### Noţiuni de adresare a memoriei

Memoria unui calculator poate fi accesată dacă se cunoaşte o adresă de memorie. În acest sens, octeţii sunt numerotaţi secvenţial: 0,1,2,3, etc. Componentele hardware lucrează cu aceste adrese de memorie, cunoscute sub numele de *adrese absolute*. Programele software folosesc însă alt tip de adresare, *adresarea relativă*. O adresă relativă reprezintă o adresă de memorie exprimată relativ la o locaţie de bază. De exemplu, o rutină a unui program poate fi scrisă astfel încât primul octet al său se află la adresa 0 (numită şi *punct de intrare*) iar celelalte adrese sunt exprimate ca adrese de deplasament (offset) relativ la această adresă. Atunci când rutina de program este încărcată în memorie, adresa absolută de început este stocată într-un registru, stabilindu-se o adresă de bază pentru rutina program.

Adresarea relativă are două scopuri: în primul rând, permite unei rutine să fie relocalizată în memorie (să poată fi încărcată într-o locaţie diferită de memorie de fiecare dată când rulează). Deoarece fiecare locaţie a rutinei este exprimată relativ la punctul de intrare, iar acesta este stocat într-un registru de bază, totdeauna este posibil să se calculeze adresa absolută prin adunarea adresei de bază cu adresa de deplasament. În al doilea rând, deoarece doar adresa de deplasament trebuie stocată în instrucţiune, sunt necesari mai puţini biţi pentru a stoca fiecare adresă, acest lucru conducând la existenţa unor instrucţiuni de mărime mai mică.

## 1.7 Limbaje de nivel înalt şi de nivel jos. Instrucţiuni

Microprocesoarele au o mulţime fixă, restrânsă, de instrucţiuni – denumită mulţimea de *instrucţiuni maşină* (sau *cod-maşină*). Cu ajutorul acestor instrucţiuni microprocesoarele pot fi programate să execute sarcinile pe care le dorim. Fiecare producător de cipuri (procesoare), fie că este vorba de Intel, IBM, Sun sau DEC proiectează şi produce microprocesoare şi computere ce posedă un *limbaj propriu* nativ – propria mulţime de instrucţiuni maşină. Acest limbaj conţine, de regulă, 100-200 de instrucţiuni diferite. Nu există în acest caz un standard universal recunoscut cum ar fi codurile ASCII sau UNICODE, spre exemplu.

În afara acestor limbaje maşină ce posedă până la câteva sute de instrucţiuni, s-au dezvoltat aşa numitele limbaje de nivel înalt (***HLL*** – High Level Language). De aceea, limbajele maşină sunt cunoscute şi sub titulatura de limbaje de nivel jos (***LLL*** – Low Level Language). Limbajele de nivel înalt sunt preferate de regulă de către programatori, deoarece acestea oferă un set de instrucţiuni şi concepte apropiate de limbajul natural. Exemple: BASIC, Pascal, C, C++, Java. Instrucţiunile unui astfel de limbaj de nivel înalt trebuie însă translatate în codul maşină de nivel jos pentru ca

microprocesorul să le înţeleagă şi să le ruleze. Această translatare expandează o instrucţiune HLL în 3-10 instrucţiuni LLL.

**Exemplu:**

| | |
|---|---|
| **HLL (C)** | i=j+k; |
| **ASM** | MOV eax,[12011234]<br>ADD eax,[12011238]<br>MOV [1201123C], eax |
| **Cod binar maşină** | 0010 0000 0011 1001<br>0001 0010 0000 0001<br>0001 0010 0011 0100<br>1101 0000 1011 1001<br>0001 0010 0000 0001<br>0001 0010 0011 1000<br>0010 0011 1100 0000<br>0001 0010 0000 0001<br>0001 0010 0011 1100 |

În primele stadii de dezvoltare ale computerelor problema de bază era cum să fie scris un program şi apoi cum să fie introdus în memoria computerului. Iniţial, programele erau scrise de mână în binar şi introduse (tot manual!) în memorie.

Instrucţiunile maşină pot fi grupate în patru mari categorii. Ele pot fi instrucţiuni pentru:

- Transferul şi manipularea datelor;
- Operaţii de intrare/ieşire;
- Transferul controlului programului;
- Controlul maşinii (termină procesarea, resetează maşina etc.).

### 1.8 Compilatoare şi asambloare

Compilatoarele translatează instrucţiunile HLL în binar cod - maşină pentru a fi direct executate de către computer. Ele au devenit componente de bază într-un sistem de calcul, reflectând, de regulă, eficienţa cu care resursele harware sunt exploatate de către programatori. Înainte de apariţia limbajelor de nivel înalt şi a compilatoarelor, programatorii foloseau limbajul de asamblare (ASM) şi asambloarele. Acest lucru avea, de asemenea, avantajul de a nu scrie programe direct în binar. Limbajul de asamblare posedă instrucţiuni cod-maşină ce poartă denumirea de

*mnemonice* (provin de la cuvintele în engleză ce specifică acţiunea lor) precum ADD, MOV, SUB sau JMP.

Programul apare sub forma unei liste de astfel de mnemonice ce pot fi convertite uşor în limbajul de cod binar. Programele ce fac această translatare se numesc asambloare. Exemple de limbaje de asamblare sunt: MASM (Microsoft Assembler) şi TASM (Turbo Assembler – dezvoltat de firma Borland).

Translatarea unui program HLL se poate face fie prin intermediul unui *compilator*, fie prin intermediul unui *interpretor*. Limbajele BASIC şi Java au pornit la început folosind interpretoare, dar acum există şi compilatoare pentru ele.

Pentru a fi executat de către microprocesor, un program trebuie să parcurgă următoarele etape (figura 1.7):

- Editare (*EDIT*)
- Compilare (*COMPILE*)
- Editare de legături (*LINK*)
- Încărcare în memorie (*LOAD*)

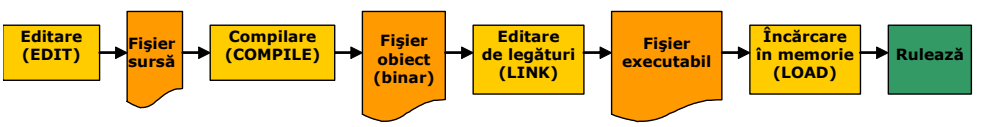



Fig. 1.7 Schema generală *EDIT-COMPILE-LINK-LOAD*

## 1.9 Editarea de legături

Dimensiunile mari ale programelor de astăzi a condus la împărţirea acestora în mai multe părţi, denumite *module*. Pentru obţinerea unui program executabil, fiecare modul trebuie proiectat, scris şi compilat iar apoi toate componentele rezultate trebuie combinate împreună de un editor de legături (*linker* sau *link editor*). Acest proces nu implică doar punerea împreună a componentelor, ci presupune rezolvarea referinţelor externe. Atunci când un program este partiţionat în module sunt frecvente cazurile în care codul dintr-un modul trebuie să facă referire la date sau subrutine dintr-un alt modul pereche. Compilatoarele translatează un singur modul la un moment dat. Atunci când un compilator încearcă să determine echivalentul numeric al acestor *referinţe simbolice*, va descoperi că nu există valori valide în cadrul modului curent. Aceste simboluri nerezolvate sunt denumite *referinţe externe* şi rămân simbolice (fără o valoare numerică) până în momentul în care editorul de legături rezolvă aceste referinţe.

## 1.10 Interpretoare

Interpretoarele oferă o alternativă pentru a rula programe scrise într-un limbaj înalt de programare. În loc de a translata instrucţiunile de nivel înalt în cod maşină şi de a crea un program executabil, interpretorul citeşte instrucţiunile de nivel înalt *una câte una* şi le execută folosind o bibliotecă proprie de rutine. În acest mod, codul executabil nu este generat din codul sursă ci este conţinut în cadrul interpretorului.

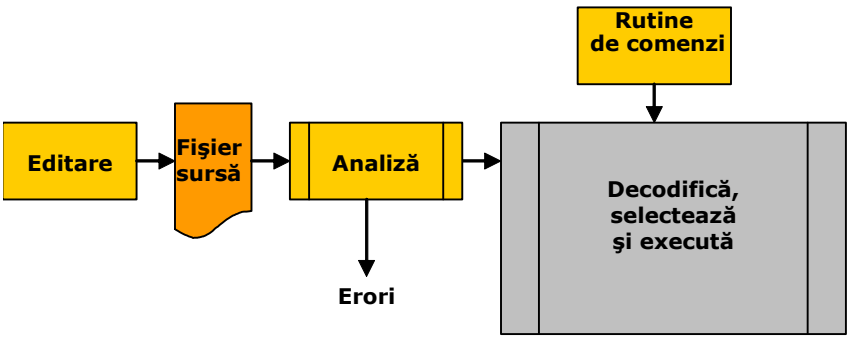



Fig. 1.8 Modul de lucru al unui interpretor

Interpretorul "vede" codul sursă HLL ca date de intrare care trebuie analizate şi apoi trebuie procesate conform semnificaţiei lor. Avantajele folosirii unui interpretor sunt: pornirea rapidă şi aparenta lipsă a complexităţii legate de compilare şi editare de legături. Dezavantajul este că operează foarte lent; programele compilate au fost întotdeauna superioare programelor interpretate.

Este ceva obişnuit ca interpretoarele să convertească instrucţiunile de intrare într-o formă intermediară alcătuită din anumite simboluri, înainte de a se lua o decizie asupra acţiunilor ce vor fi executate (figura 1.8). Simbolurile generate sunt apoi trecute unui decodificator care va selecta rutina corespunzătoare ce va fi executată. Uneori interpretorul este văzut ca o "maşină virtuală" deoarece se comportă într-un fel ca o componentă hardware: citeşte instrucţiuni pe rând şi se supune lor. Interpretoarele apropie nivelul execuţiei instrucţiunilor de nivelul programului conceput în limbajul de nivel înalt. Limbajul de programare Java posedă o modalitate interesantă de compilare şi interpretare ilustrată în figura 1.9.

## 1.11 Monoprocesare şi multiprocesare. Definiţii

În continuare prezentăm câteva definiţii legate de mono-procesare, multi-procesare, execuţie paralelă, procese şi fire de execuţie, noţiuni necesare pentru înţelegerea cunoştinţelor prezentate în continuare.

**Calculator monoprocesor**

Prin definiţie, un calculator dotat cu o singură unitate centrală de procesare (microprocesor) se numeşte calculator mono-procesor.

**Calculator multiprocesor**

Un calculator multiprocesor este un calculator ce conţine două sau mai multe UCP.

**Task**

Un *task* reprezintă o mulţime de instrucţiuni executate de către un procesor. Din acest punct de vedere, definiţia este echivalentă cu aceea de aplicaţie sau program.

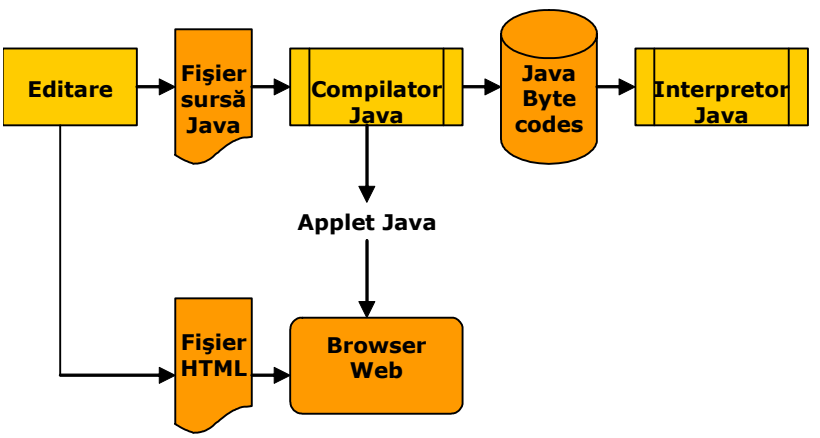



Fig. 1.9 Compilarea şi interpretarea Java

**Sistem de operare monotasking**

Un sistem de operare este un sistem monotasking dacă poate executa o singură aplicaţie (sau *task*) într-un anumit moment. Sistemul de operare MS-DOS este un exemplu de sistem de operare monotasking. Sub sistemul de operare MS-DOS un program poate fi executat prin introducerea numelui său la linia de comandă. Până când programul curent nu se termină, nu se poate lansa în execuţie alt program.

**Sistem de operare multitasking**

Un sistem de operare multitasking poate executa mai multe aplicaţii (task-uri) în acelaşi timp. Acest lucru este posibil prin execuţia alternativă a unor segmente din cele două (sau mai multe) task-uri. Împărţirea task-urilor în segmente şi execuţia alternativă a acestor segmente poartă denumirea de *time-slicing*. Există două tipuri de sisteme de operare multitasking: *cooperative* şi *preemptive*.

Un sistem de operare *cooperativ* se bazează pe capacitatea aplicaţiilor ce se execută pentru implementarea multitasking-ului. Aplicaţia respectivă trebuie să facă un apel (de sistem) către sistemul de operare, întrerupându-şi execuţia şi permiţând altei aplicaţii să ruleze. Dacă aplicaţia nu face acest apel, va prelua controlul complet asupra procesorului, făcând imposibilă execuţia altui program. Un exemplu de sistem de operare multitasking cooperativ este Windows 3.1.

Un sistem de operare *preemptiv* nu are nevoie ca aplicaţiile să emită apeluri de sistem pentru a permite execuţia şi a altor programe. Modalitatea de control a sistemului de operare este alta: fiecărei aplicaţii i se alocă un timp pentru a rula; în momentul în care acest timp a expirat, aplicaţia este automat întreruptă şi se permite altei aplicaţii să utilizeze capacitatea de calcul a microprocesorului. Majoritatea sistemelor de operare de astăzi sunt sisteme multitasking preemptive (UNIX, Linux, toate variantele de Windows după Windows NT inclusiv).

**Execuţie în paralel**

Execuţia în paralel (sau paralelă) reprezintă execuţia simultană a două sau mai multe task-uri pe un calculator multiprocesor.

**Proces**

Un proces poate fi definit ca o instanţă a unui program aflat în execuţie. Fiecare proces posedă o mulţime de instrucţiuni denumită "segment de text" ce defineşte modalitatea de funcţionare a procesului şi segmentul de memorie ce este folosit pentru execuţia acestuia. Segmentul de text conţine o instrucţiune specifică prin care se specifică unde va începe să se execute procesul. Memoria utilizată de către un proces este accesibilă doar acelui proces. Oricare alt proces are interzis accesul în acest spaţiu de memorie.

**Thread**

Un *thread* (fir de execuţie) reprezintă un flux de execuţie în cadrul unui proces; astfel, un singur proces poate conţine mai multe fire de execuţie. Sistemul de operare asigură controlul acestor fire de execuţie şi alocă timpi de execuţie (de către microprocesor) pentru ca acestea să se execute.

**Multithreading**

Operaţiunea de multithreading este echivalentă cu multitasking-ul la nivelul unui singur proces. Acest termen descrie posibilitatea ca o aplicaţie

ce se execută pe un calculator să aibă mai multe fire de execuţie. Un proces ce este împărţit în mai multe fire de execuţie are capacitatea de a executa mai multe seturi de instrucţiuni în mod concurent. Diferenţa majoră între execuţia mai multor fire de execuţie în cazul unui proces şi mai multe procese ce se execută pe un sistem este aceea că firele de execuţie partajează acelaşi spaţiu de memorie în cadrul aceluiaşi proces, pe când procesele separate au spaţii de memorie distincte.

**Multiprocesarea**

Multiprocesarea poate avea mai multe definiţii în funcţie de contextul de referinţă. Având în vedere definiţia unui proces, o definiţie logică a multi-procesării este aceea a capacităţii unui calculator de a executa mai multe procese simultan. Din această perspectivă, definiţia coincide deci cu definiţia multitasking-ului.

Din punct de vedere al capacităţii fizice a unui calculator de a executa mai multe instrucţiuni în aceeaşi perioadă de timp, nu ne putem referi decât la calculatoare multiprocesor. În acest caz, definiţia multiprocesării coincide cu definiţia execuţiei în paralel.

Există şi o a treia semnificaţie a multiprocesării, şi anume când se priveşte din perspectiva unei anumite aplicaţii software. Dacă o aplicaţie suportă multiprocesarea, atunci pot fi executate mai multe instanţe ale acelei aplicaţii în mod concurent.

## 1.12 Rezumat

Un calculator reprezintă un dispozitiv complex, a cărui cunoaştere şi înţelegere a principiilor de funcţionare implică modularizarea studiului pe mai multe nivele structurale. Calculatorul reprezintă o conclucrare între componentele fizice (hardware) şi programe (componenta software). Sistemul de operare asigură interfaţa între componenta hardware a sistemului de calcul şi programele de aplicaţii. De asemenea, sistemul de operare cuprinde o colecţie (partajată de celelalte programe de aplicaţii) ce cuprinde rutine de acces hardware şi defineşte o platformă pentru elaborarea de software de aplicaţie.

Un sistem de operare modern cuprinde o serie de rutine ce asigură următoarele funcţionalităţi de bază: interfaţa cu utilizatorul (shell-ul), managementul fişierelor (sistemul de fişiere), managementul procesorului, managementul dispozitivelor periferice şi managementul memoriei.

Componentele de bază ale unui calculator sunt: unitatea centrală de procesare (UCP - microprocesorul), memoria principală, dispozitivele de intrare/ieşire şi magistrala de interconexiune a componentelor. În continuare vom prezenta detalii referitoare la aceste principale elemente arhitecturale ale unui calculator.

# 2

# MICROPROCESORUL

## 2.1 Microprocesorul, componenta principală a calculatorului

Microprocesorul reprezintă unitatea centrală de procesare (UCP) a unui calculator, componentă ultracompactă şi de o importanță majoră; alegerea acesteia determină limitele performanței şi prețul sistemului de calcul. Pentru a înțelege performanțele şi capacitățile unui calculator, trebuie cunoscute opțiunile existente în materie de microprocesoare. În zilele noastre, majoritatea microprocesoarelor care rulează software pentru calculatoare personale (PC - Personal Computer) sunt variante îmbunătățite ale vechiului procesor 8086 (procesor care în 2003 a împlinit 25 de ani de la apariție!) şi au la bază arhitectura x86.

Datorită popularității acestui tip arhitectural, chiar şi unele procesoare cu o arhitectură diferită (RISC - Reduced Instruction Set Computer) pot rula software pentru PC, emulând funcțiunile procesoarelor Intel x86, sau încorporând unități de execuție a instrucțiunilor compatibile cu cele de la Intel. De remarcat este faptul că, datorită superiorității evidente a procesoarelor RISC, chiar Intel a preluat pentru ultimile modele de procesoare concepte RISC, păstrând însă compatibilitatea cu vechea arhitectură x86 (Procesoarele Pentium Pro, Pentium II, Pentium III şi Pentium 4, chiar şi vechiul Pentium). Dacă la începutul anilor 80 unii ingineri de produse RISC ridiculizau familia x86 şi îi prevedeau un sfârşit apropiat, iată că Intel a contrazis aceste previziuni şi, mai mult, confruntarea RISC-CISC s-ar putea foarte probabil să aibă un deznodământ contrar celui anticipat acum 20 de ani. Acesta este un exemplu ilustrativ în sensul că afacerile, pe de o parte, şi tehnologia, pe de alta, sunt două lucruri diferite. Cu toate că tehnologia RISC este superioară tehnic celei CISC, vastele

resurse ale firmei Intel şi succesul prelungit al sistemelor de operare DOS şi Windows au menţinut vechea arhitectură x86 în competiţie.

Mai mult, acum situaţia s-a inversat: noua arhitectură IA-64 elaborată în parteneriat de către Hewlett-Packard şi Intel, ar putea duce la declinul tehnologiei RISC. Arhitectura IA-64 reprezintă o combinaţie de caracteristici inovatoare precum: execuţie speculativă, paralelism îmbunătăţit, stivă de registre, arhitectură superioară de ramificare şi nu numai. Modalitatea de adresare a memoriei a fost îmbunătăţită pentru a satisface cerinţele tot mai mari de memorie ale noilor aplicaţii de e-business, depozite de date *(data warehousing)* sau a altor aplicaţii performante pentru servere. Arhitectura IA-64 posedă şi o arhitectură în virgulă mobilă superioară şi alte modificări în scopul de a oferi suport aplicaţiilor staţiilor de lucru de înaltă performanţă legate de crearea de conţinut digital, proiectare inginerească sau analiză ştiinţifică.

Arhitectura IA-64 se doreşte a fi extrem de scalabilă pentru a satisface cerinţele de creştere a performanţelor atât pe segmentul de piaţă destinat staţiilor de lucru cât şi pe cel al serverelor. Arhitectura IA-64 reprezintă un concept revoluţionar pe 64 de biţi *(ISA – Instruction Set Architecture)* ce aplică o arhitectură tehnologică nouă, denumită *EPIC (Explicit Parallel Instruction Computing)*. De asemenea, un element de bază pentru IA-64 este acela al compatibilităţii cu mulţimea de instrucţiuni IA-32.

Procesoarele sunt dispozitive complicate, incredibil de puternice, dar performanţele şi proprietăţile acestora din zilele de azi sunt rezultatul unor ani de evoluţie şi îmbunătăţiri plecând de la nişte începuturi umile.

Primele dispozitive de calcul erau electromecanice, folosind schimbătoare de stare fizice. Acestea erau foarte încete, nesigure şi zgomotoase datorită componentelor mecanice, ceea ce însemna că ceva se mişca atunci când un bit îşi schimba valoarea. Următoarea generaţie de dispozitive folosea tuburi cu vacuum. Acestea reprezentau o îmbunătăţire majoră, în sensul că nu mai aveau părţi mişcătoare. Tuburile electronice au permis crearea primelor calculatoare electronice, care aveau însă multiple probleme în funcţionare.

O dată cu inventarea tranzistorului în 1948 la Bell Laboratories (SUA), tehnologia s-a îmbunătăţit, tranzistorii fiind mult mai mici, mai rapizi şi mai ieftini. În continuare, faptul care a permis crearea procesoarelor moderne a fost inventarea circuitului integrat, care este un grup de tranzistori fabricaţi dintr-o singură bucată de material şi conectaţi intern,

fără cablaje externe. Circuitele integrate se mai numesc prescurtat CI sau cipuri.

Dintre toate materialele ce conduc curentul electric, doar unele pot să facă acest lucru în anumite condiții. Astfel au apărut semiconductorii. Cel mai folosit semiconductor este siliciul. A apărut astfel tehnologia implantului pe o pastilă de siliciu a mai multor tranzistori ce sunt rapizi, siguri şi folosesc relativ puțină putere. Primul circuit integrat a fost inventat în 1959 de Texas Instruments. El conținea doar şase tranzistori pe o suprafață semiconductoare.

După inventarea circuitului integrat, a durat puțin timp pentru a conştientiza imensele beneficii ale miniaturizării şi integrării unui număr tot mai mare de tranzistori în acelaşi cip. Tehnica LSI - Large Scale Integration - a însemnat creearea circuitelor integrate ce au fost construite la început din mai multe componente discrete. Aceste dispozitive conțineau de regulă sute de tranzistori. Primele computere au fost realizate încorporând mai multe circuite integrate mici conectate pe circuitul de pe placă. O imagine a unui circuit integrat pe bază de siliciu este prezentată în figura 2.1.

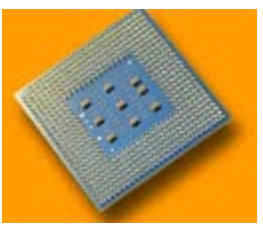


Fig. 2.1 Circuit integrat - microprocesor

O dată cu trecerea timpului, tehnologia LSI s-a îmbunătățit şi cipurile au devenit mai mici, mai rapide şi mai ieftine. Bazându-se pe succesul anterior, inginerii au învățat să încorporeze mai multe funcții logice într-un singur circuit. A apărut astfel VLSI - Very Large Scale Integration - tehnologie ce înglobează milioane de tranzistori. La început, funcțiile executate de procesor erau implementate folosind cipuri logice diferite. Intel a fost prima companie ce a încorporat toate aceste componente logice într-un singur cip. Acesta a fost primul microprocesor, 4004, produs în 1971. Toate microprocesoarele de astăzi sunt descendenți (unii foarte avansați) ai acestui prim cip pe 4 biți.

În viitorul apropiat tehnologia VLSI va deveni tehnologie GSI (GigaScale Integration) şi chiar TSI (TeraScale Integration). În acest context, *gigascale* şi *terascale* semnifică existența a mai mult de un miliard de cipuri, respectiv un milion de miliarde de componente integrate într-un singur cip. Din perspectiva proiectării de sistem această creştere fantastică a nivelului de integrare este calitativ diferită față de îmbunătățirile apărute în trecut. În particular, numărul defectelor de manufacturare vor creşte, echipamentele vor deveni mai puțin fiabile, vor apărea noi nano-tehnologii iar problemele de administrare a puterii şi a semnalelor se vor complica. Este foarte probabil ca noile nano-tehnologii să complementeze sau chiar să înlocuiască tehnologia CMOS. În zilele noastre, tehnologia CMOS (Complementary Metal-Oxide Semiconductor) reprezintă tehnologia dominantă pe piața semiconductorilor pentru microprocesoare, memorii şi circuite integrate specializate pe aplicații *(ASIC – Application Specific Integrated Circuits)*.

## 2.2 Ciclul fetch-decode-execute

Funcționarea calculatorului implică în fiecare moment anumite activități prin care acesta execută diverse comenzi primite de la sistemul de operare sau de la programele utilitare, comenzi ce implică de fiecare dată execuția unor instrucțiuni. Operația de bază a unui procesor este reprezentată de aşa-numitul ciclu "Fetch-Decode-Execute" sau, pe scurt, "Fetch-Execute". Pe scurt, această operație implică faptul că pentru a executa o instrucțiune, de fiecare dată, procesorul citeşte o instrucțiune de program din memorie, o decodifică şi apoi o execută. Acest lucru poate părea anevoios dar viteza fantastică la care acționează microprocesorul face ca procedeul de aducere din memorie, decodificare şi apoi execuție să se desfăşoare extrem de rapid.

Cei trei „actori" implicați în acest proces, microprocesorul, memoria şi magistrala de memorie acționează la diferite viteze, de aceea, cu cât una dintre componente are o viteză de acționare mai mare, cu atât va contribui mai mult la scurtarea timpului de execuție a unui program. Există aplicații ce solicită intens lucrul cu memoria – în acest caz memoriile ieftine şi rapide prezintă un avantaj, în timp ce alte aplicații pot solicita intens activitatea procesorului – desigur că în acest caz un procesor mai rapid va creşte performanțele de calcul.

Din punct de vedere al tipului de memorie principală folosit de către calculatoare, există în principiu două tipuri de bază: cipuri DRAM (Dynamic Random Access Memory), care nu sunt atât de rapide precum procesorul. Un alt tip de memorie este memoria SRAM (Static Random Access Memory), care funcționează la viteze mai mari dar are şi costuri de producție mult mai ridicate; astfel de cipuri de memorii sunt utilizate la memoriile *cache*. Memoriile cache sunt utilizate pentru a reduce decalajul dintre viteza microprocesorului şi aceea a memoriei principale prin copierea şi stocarea instrucțiunilor şi datelor ce vor fi folosite imediat de către procesor.

Procesoarele RISC şi chiar procesoarele CISC (Intel) au introdus o serie de tehnici noi pentru a mări eficiența ciclului „Fetch-Execute" prin procesarea simultană a mai multor instrucțiuni; cea mai cunoscută astfel de tehnică fiind tehnica prelucrării în conductă *(pipeline)* a instrucțiunilor.

Computerul trebuie să citească şi să se supună fiecărui program (inclusiv sistemului de operare), instrucțiune cu instrucțiune. La prima vedere este un handicap din punct de vedere al performanțelor dacă este comparat cu organismele vii capabile să desfăşoare mii de activități simultan. Operația de bază pe care o îndeplineşte procesorul este operația fetch-execute, secvență în care fiecare instrucțiune din cadrul unui program este citită din memorie asociată programului în CPU, este decodificată şi apoi executată. Doar viteza extraordinară a echipamentelor electronice face ca acest ciclu obositor care se repetă încontinuu să fie de reală valoare.

În orice fază a dezvoltării computerelor, una dintre cele trei unități implicate în acest ciclu – memoria, magistrala şi CPU a fost factorul limitativ. Acest lucru poate afecta atât parametrii cu care trebuie să lucreze inginerii şi, de asemenea, selectarea algoritmilor pentru rezolvarea problemelor. De exemplu, există uneori disponibile metode care solicită mai mult memoria (*memory-intensive*) sau alteori metode care solicită mai mult capacitatea de procesare (*compute-intensive*).

Dacă memoria este rapidă şi ieftină, atunci primele metode sunt de preferat; în celălalt caz este preferată a doua soluție. În prezent, cipurile DRAM de memorie nu sunt la fel de rapide precum CPU. Există disponibile şi cipuri mai rapide SRAM, dar care sunt mult mai scumpe şi de aceea sunt utilizate în mici *buffere* foarte rapide denumite şi memorii *cache*. Acest tip de memorii poate ajuta într-o oarecare măsură la reducerea diferenței dintre întârzierile cauzate de accesul la memorie prin stocarea unor cópii ale instrucțiunilor şi datelor curente.

Pentru a reduce şi mai mult gâtuirea obținută datorată efectului negativ a structurii secvențiale von Neumann, noile procesoare RISC măresc viteza de execuție a ciclului fetch-execute prin execuția simultană a (~5) instrucțiuni prin intermediul unei metode pipeline.

Oricât de familiari am fi cu modalitatea de lucru a computerelor, este practic imposibil să înțelegem viteza incredibilă la care acestea operează. Tabelul următor (tabelul 2.2) ne prezintă diferențele între viteze ale operațiunilor uzuale din lumea reală.

Tabelul 2.2

**Exemple de activități şi viteza asociată acestora**

| **ns= 1 / 1000 000 000 s** | **μs = 1 / 1 000 000 s** | **ms = 1 / 1 000 s** |
|---|---|---|
| Ciclul fetch/execute – 10 ns | Viteza luminii – 300 m/μs | Viteza de reacție umană – 300 ms |
| Funcționarea unei porți logice – 5 ns | Linie de scan TV – 60 μs | Cadru TV – 20 ms |
| Acces la memoria SRAM – 15 ns | Întrerupere – 2-10 μs | Acces la hard-disk – 10 ms |
| | Scânteie la motor – 10 μs | Rotație completă a motorului la maşină (la 3000 rpm) – 20 ms |

Din tabel ne putem da seama de diferența existentă între viteza de operare a calculatorului şi diferite viteze din lumea înconjurătoare. Lucrurile ce par extrem de rapide, cum ar fi liniile de scan TV, sunt de sute de ori mai încete decât un ciclu fetch-execute realizat de către microprocesor. După cum ştim, la cel mai de jos nivel programele în calculator sunt formate din şiruri de biți ce reprezintă codificarea binară a unor instrucțiuni, precum:

**1000 1011 1110 1100** Acest şir de biți este echivalent cu următoarea instrucțiune a procesorului Pentium:

**MOV BP , SP**

Rezultatul instrucțiunii anterioare este acela de a copia conținutul registrului SP în registrul BP.

De asemenea, şirul de biţi:

**1000 1011 0011 0100 0001 0010**

este echivalent cu instrucţiunea următoare în limbaj de asamblare (pentru Pentium):

MOV AX, 1234H

Sau, în limbajul C:

**x = 4660;**

Se observă astfel avantajul unui limbaj de programare de nivel înalt (în cazul nostru limbajul C) faţă de exprimarea în limbaj cod-maşină (şiruri de biţi) sau în limbaj de asamblare. Instrucţiunea de mai sus va copia valoarea zecimală 4660 (*1234* în hexazecimal) în registrul AX (denumit şi registrul *acumulator*).

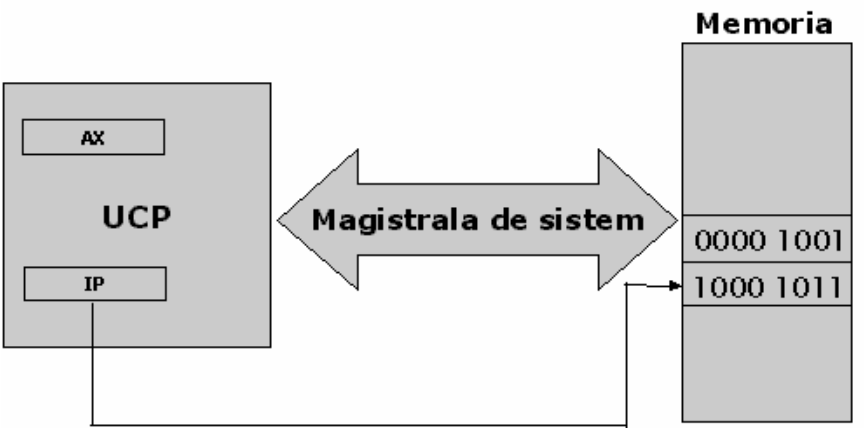



Fig. 2.3. IP (Pointerul de instrucţiune) indică întotdeauna adresa următoarei instrucţiuni ce va fi executată

Ciclul fetch-execute reprezintă procesul prin care microprocesorul preia din memoria în care este stocată programul următoarea instrucţiune ce va fi executată, o decodifică şi execută operaţia pe care această instrucţiune o reprezintă.

În continuare prezentăm funcţionarea generală a acestui proces. În figura 2.3 putem vedea microprocesorul cu registrul pointer de instrucţiune - IP *(Instruction Pointer)* şi registrul acumulator AX *(Accumulator Register).* Memoria principală stochează programul aflat în execuţie, în care toate instrucţiunile de genul: **MOV BP,SP** sau **MOV AX, 1234H** se află reprezentate sub formă binară. Un registru de acces la memorie este utilizat

pentru a putea accesa porțiunea de memorie de unde sunt preluate datele. Registrul pointer de instrucțiune indică tot timpul adresa din memorie a următoarei instrucțiuni ce va fi executată.

Figurile 2.4 şi 2.5 ne înfățişează procesul de preluare a datelor din memorie (etapa *fetch*). Această etapă este aproximativ identică pentru toate tipurile de instrucțiuni. Paşii urmați de microprocesor în această etapă sunt:

**a)** adresa din registrul pointer de instrucțiune (IP) este copiată pe magistrala de adrese de memorie de unde este transmisă în registrul RAM (Registrul de Acces la Memorie);

**b)** pointerul de instrucțiune este incrementat (IP++), indicând adresa de memorie a următoarei instrucțiuni ce va fi executată;

**c)** se selectează locația de memorie şi se copiază conținutul acesteia în magistrala de date;

**d)** procesorul copiază codul instrucțiunii din magistrala de date în registrul de instrucțiune;

**e)** începe procesul de decodificare a instrucțiunii.

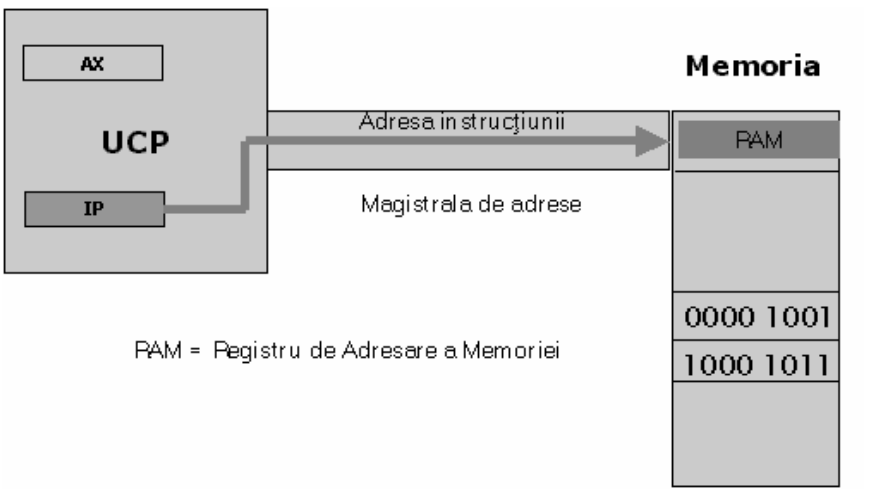



Fig. 2.4 Ilustrarea pasului a) din ciclul *fetch*

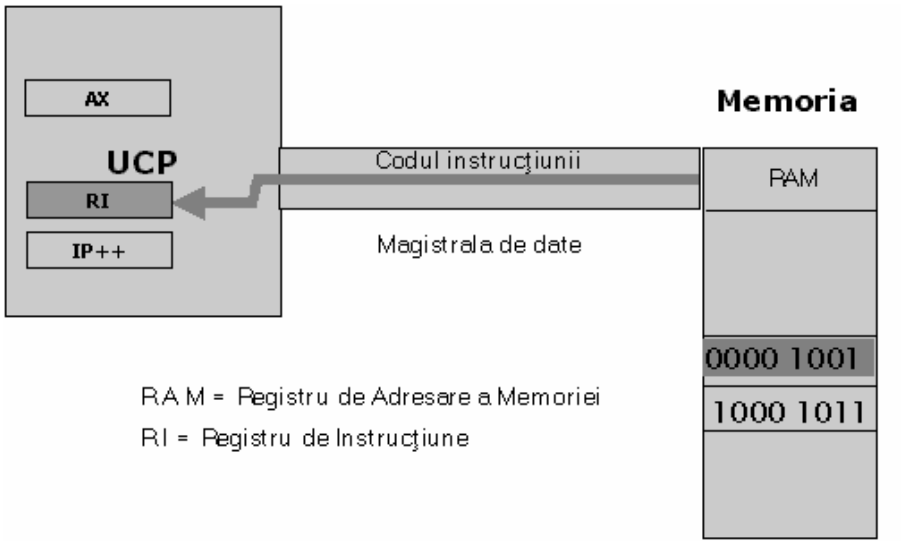



Fig. 2.5 Ilustrarea paşilor b), c) şi d) din ciclul f*etch*

Figurile 2.6 şi 2.7 ne înfăţişează procesul de execuţie a instrucţiunii (etapa *execute*) în cazul instrucţiunii Intel **MOV AX, 1234H** (etapa *execute* diferă de la instrucţiune la instrucţiune). Paşii urmaţi de microprocesor în această etapă sunt:

**a)** conţinutul registrului pointer de instrucţiune (IP) este copiat pe magistrala de adrese de memorie de unde este transmis în registrul RAM (Registrul de Acces la Memorie);

**b)** pointerul de instrucţiune este incrementat (IP++);

**c)** valoarea selectată din memorie (1234H) este copiată pe magistrala de date;

**d)** procesorul copiază valoarea de pe magistrala de date în registrul AX.

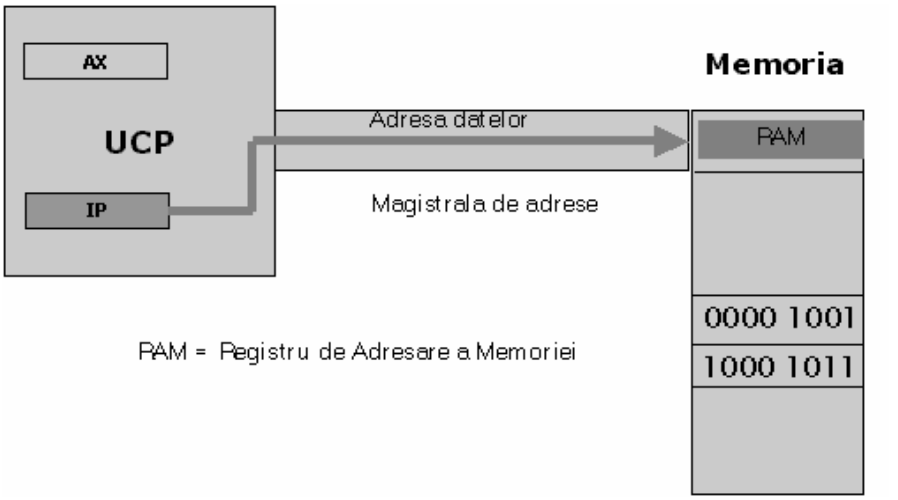



Fig. 2.6 Ilustrarea pasului a) din ciclul *execute*

Acest proces reprezintă, de fapt, o simplificare a procesului *fetch-execute* ce se desfăşoară în cadrul unui microprocesor modern din zilele noastre. Unele instrucţiuni au nevoie de un ciclu de execuţie în plus pentru a citi valoarea unei adrese din memorie, care este mai apoi folosită pentru a accesa valoarea variabilei respective, valoare stocată tot în memorie. Ca o concluzie, ciclul fetch-execute reprezintă secvenţa prin care fiecare instrucţiune a unui program este citită din memorie, decodificată şi apoi executată. Acest proces poate presupune ulterior mai multe subprocese, cum ar fi citirea de date suplimentare din memorie şi stocarea rezultatelor operaţiei/operaţiilor înapoi în memorie.

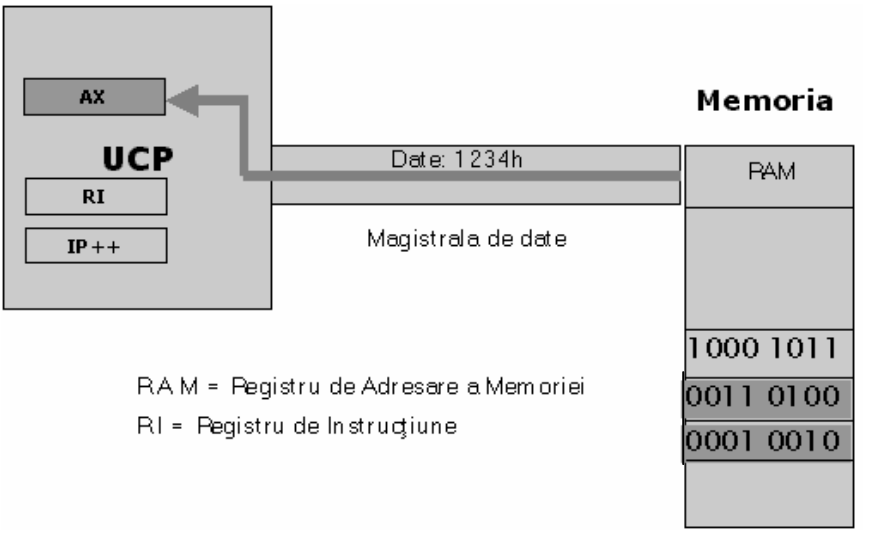



Fig. 2.7 Ilustrarea paşilor b), c) şi d) din ciclul *execute*

Atât sistemul de operare Windows NT/2000 cât şi sistemul de operare UNIX oferă instrumente prin care pot fi vizualizate activităţile ce se află în lucru. Pe staţiile Sun cu sistemul de operare UNIX (varianta Solaris) există utilitarul denumit *perfmeter*, pentru Linux există *xsysinfo şi gsysinfo* iar pentru Windows NT există *Performance Monitor,* care a fost înlocuit în Windows 2000 de *System Monitor*. Prezentăm în continuare caracteristicile acestor aplicaţii utilizate pentru monitorizarea utilizării resurselor calculatorului.

- **UNIX – *perfmeter***

Pentru o staţie Sun, comanda care ne arată detaliat activitatea microprocesorului este:

$ perfmeter –t cpu &

- **Linux**

***xsysinfo***

Conform paginii de manual, utilitarul *xsysinfo* afişează parametrii kernelului sub formă grafică. Sintaxa completă este următoarea:

**xsysinfo [-help] [-update *n*] [-[no]title] [-[no]labels] [-[no]loadavg] [-[no]load] [-[no]mem] [-[no]swap]**

Pe scurt, *xsysinfo* este o aplicaţie XWindow folosită pentru afişarea unor parametri ai kernelului Linux în format grafic, o combinaţie a

comenzilor *top*, *free* şi *xload*, cu diferența că valorile afişate (media gradului de utilizare a procesorului, gradul de utilizare a procesorului, dimensiunea de *swap*) sunt prezentate într-o fereastră orizontală.

Utilitarul *xsysinfo* afişează următoarele valori:

- *gradul mediu de utilizare al UCP* – valoarea afişată este între 0.000 şi 8.000. Bara orizontală a afişajului este împărțită în segmente, unde fiecare segment reprezintă o valoare de 1.

- *gradul de utilizare al UCP* – în cazul unui sistem multiprocesor opțiunea *–smp* afişează câte o bară orizontală pentru fiecare procesor în parte.

- *memoria* – bara grafică ce corespunde memoriei este împărțită în două segmente ce reprezintă dimensiunea fizică a memoriei calculatorului ce este utilizată de procese (în partea stângă) şi memoria utilizată pentru paginare şi memoria cache buffer în partea dreaptă. Întreaga lungime a segmentului orizontal ne arată memoria fizică utilizată de către sistem la momentul respectiv.

- *porțiunea de swap* – indică dimensiunea de spațiu swap utilizat de către sistem din totalul spațiului swap alocat.

***Gsysinfo***

Gsysinfo este un utilitar conceput pentru interfața grafică Gnome şi urmăreşte utilitatea programului xsysinfo. Gsysinfo este conceput sub licență GNU – GPL (General Public Licence).

Referitor la numele de GNU, acesta provine de la sintagma „GNU Not UNIX" şi s-a dorit a fi un sistem de operare precum UNIX ce este distribuit cu codul sursă şi poate fi copiat, modificat şi redistribuit. Proiectul GNU a fost inițiat în 1983 de Richard Stallman şi alții ce au pus bazele Fundației pentru Software Liber *(FSF – Free Software Foundation)*. Concepția lui Stallman este aceea că utilizatorii pot face ce doresc cu software-ul achiziționat, putând face cópii ale acestuia pentru prieteni şi modifica codul sursă redistribuind-ul la un anumit cost. FSF stipulează termenul *copyleft* care înseamnă că oricine redistribuie software *free* trebuie să lase în continuare libertatea de copiere şi redistribuție a programului, asigurându-se în acest fel că nimeni nu va reclama drepturi de proprietate asupra unor versiuni viitoare şi nu va impune restricții la utilizarea acestuia.

În acest context, termenul *free* înseamnă *libertate* şi nu neapărat *gratis*. Fundaţia FSF percepe nişte costuri iniţiale la distribuţia GNU. Redistribuitorii pot, de asemenea, să perceapă taxe pentru copiile programelor în scopul profitului sau pentru acoperirea costurilor. Ideea de bază a *software-ului liber* (*free software*) este aceea că se lasă libertatea utilizatorilor să modifice şi să reasambleze produsul fără nici o restricţie în afară de aceea că nici ei, la rândul lor, nu pot impune restricţii mai departe.

Stallman crede că unul dintre rezultatele filozofiei *free software* este acela că mai multe programe *free* vor coexista împreună provenind din alte programe *free*. GNU• este un exemplu în acest sens; acesta a devenit un sistem de operare când în august 1996 i-a fost adăugat un kernel (GNU Hurd şi Mach). Fundaţia FSF continuă să dezvolte software *free* sub formă de programe de aplicaţii; un program de tip *spreadsheet* este acum disponibil. Sistemul de operare Linux este conceput cu componente GNU iar kernelul este dezvoltat de Linus Torvalds.

### Capturi de ecran *gsysinfo*

Prezentăm în continuare câteva capturi de ecran gsysinfo.

- putem vedea în imaginea de mai jos (figura 2.8) utilitarul gsysinfo situat între bara de volum şi ceas (încărcarea sistemului este de aproximativ 1.8):

Fig. 2.8 Ilustrare grafică a utilitarului *gsysinfo*

• GNU s-a vrut iniţial să fie o alternativă la versiunile comerciale de UNIX. Acest lucru nu s-a întâmplat încă, dar Richard Stallman şi alţi programatori muncesc în continuare pentru acest ideal. Paradoxal este că primele succese înregistrate de GNU au fost aplicaţii adiţionale sistemelor proprietare UNIX. Componente GNU precum GNU Emacs, GCC (GNU C Compiler) şi bash (un înlocuitor *free* pentru Bourne Shell) sunt instalate astăzi implicit pe majoritatea variantelor de UNIX existente.

- în continuare (figura 2.9) apare indicatorul sysinfo pentru activitatea în rețea. Traficul de date transmise de la calculator spre rețea apare în culoarea verde iar traficul de intrare este ilustrat în roşu.

Fig. 2.9 Activitatea de rețea la *gsysinfo*

- în figurile 2.10, 2.11 şi 2.12 apar ferestrele legate de setarea caracteristicilor programului *gsysinfo*.

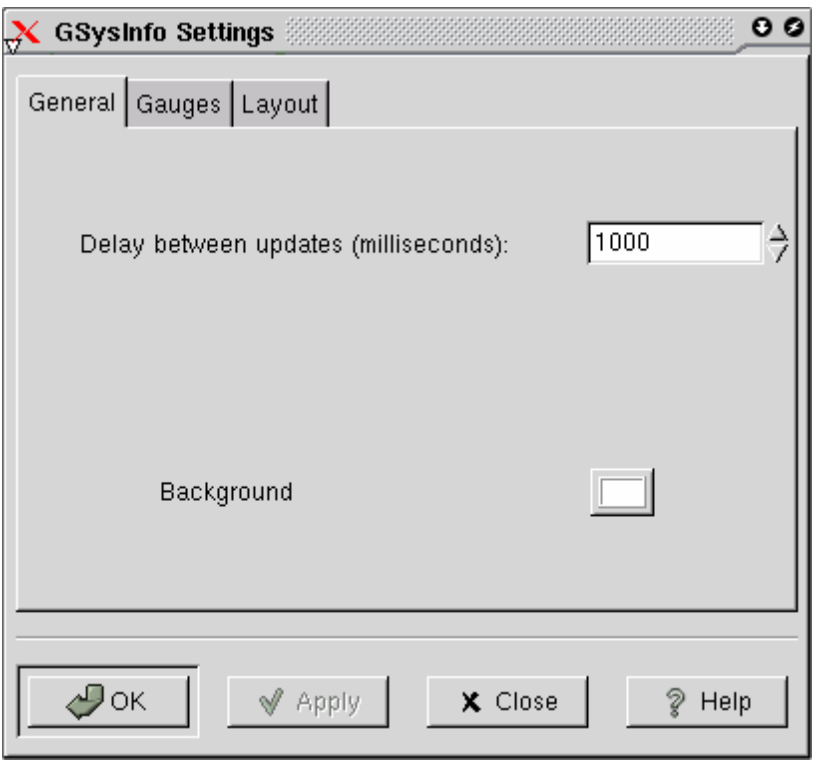



Fig. 2.10 Fereastra de proprietăți generale *gsysinfo*

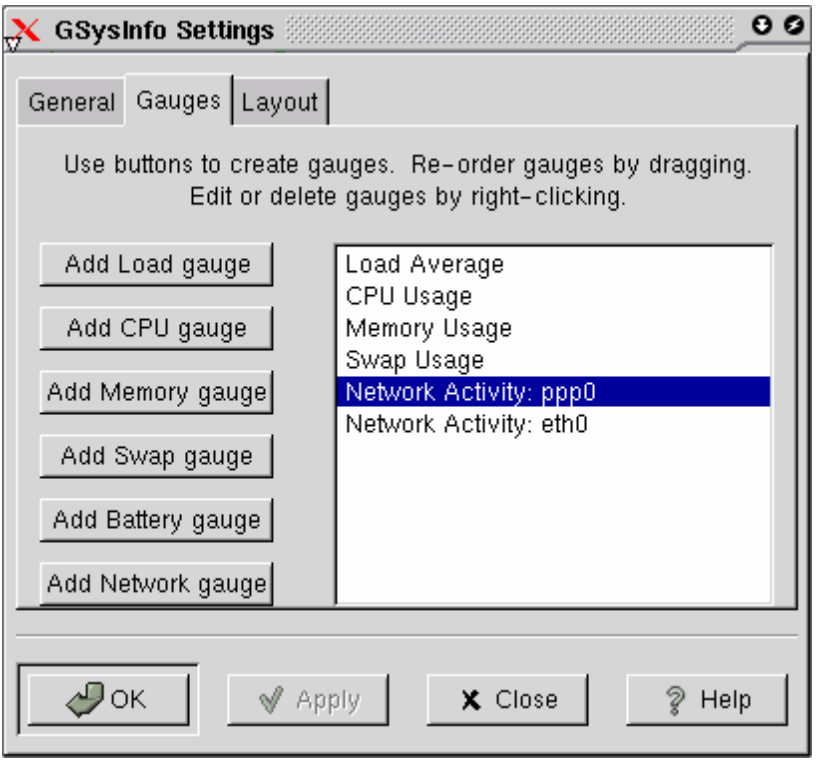



Fig. 2.11 Fereastra de setări pentru *gsysinfo* – indicatori

**Windows 2000 - System Monitor**

În Windows 2000, cu ajutorul lui System Monitor se pot măsura performanțele calculatorului local sau ale altor calculatoare din rețea. Utilitarul System Monitor asigură următoarele funcționalități:

- Colecționează şi vizualizează în timp real date legate de performanța calculatorului local sau pentru alte calculatoare de la distanță;
- Vizualizează datele colectate în timp real sau stocate anterior;
- Reprezintă datele sub formă de: grafic, histogramă sau raport de vizualizare;

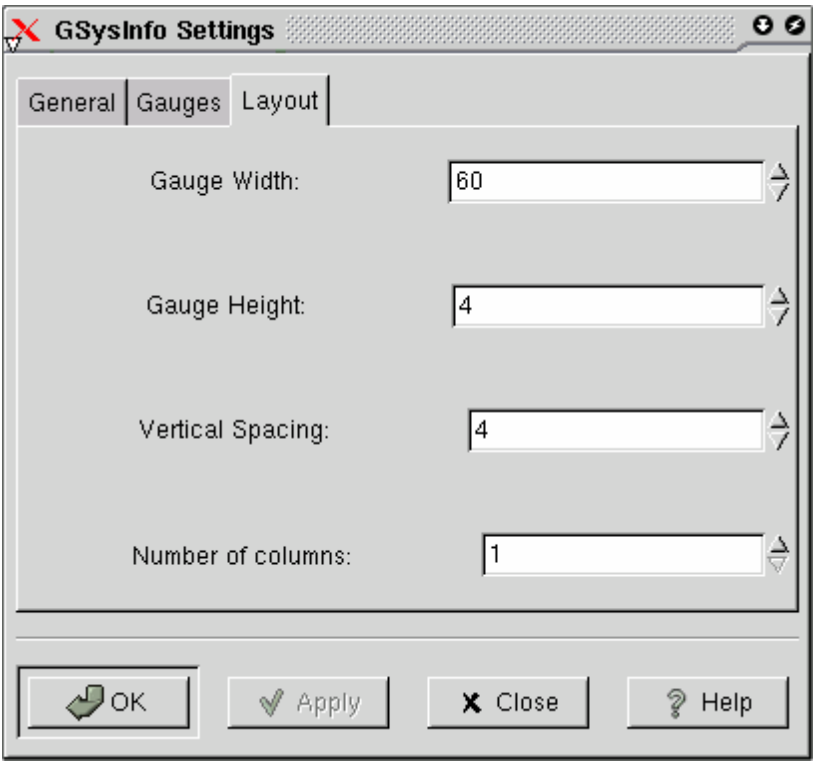



Fig. 2.12 Fereastra de setări pentru *gsysinfo* – Layout

- Încorporează funcţionalităţi ale aplicaţiei System Monitor în Microsoft Word sau alte aplicaţii ale suitei Microsoft Office cu ajutorul caracteristicei denumite *Automation*;
- Creează pagini HTML pentru vizualizarea performanţelor;
- Creează configuraţii de monitorizare reutilizabile ce pot fi instalate pe alte calculatoare ce folosesc MMC (Microsoft Management Console).

Cu ajutorul lui System Monitor se pot colecta şi vizualiza date legate de gradul de utilizare a componentelor hardware precum şi date legate de activităţile serviciilor de sistem existente pe calculatoarele administrate. În cadrul aplicaţiei se poate stabili modalitatea de prezentare a datelor în următoarele moduri:

- *Tipul de date* – pentru a selecta datele ce vor fi colectate, se pot specifica unul sau mai multe instanţe de contorizare pentru obiecte ale monitorului de performanţe. Unele obiecte

(memoria, spre exemplu) oferă contorizare la nivelul resurselor sistemului; altele oferă posibilitate de contorizare la nivelul execuției aplicațiilor.

- *Sursa de date* – Aplicația System Monitor poate strânge date de pe calculatorul local sau de pe alte calculatoare din rețea unde există această permisiune (implicit este nevoie de drepturi de administrator). În plus, se pot include atât date culese în timp real cât şi date stocate anterior în fişiere speciale de tip *log*.
- *Parametri de test* – se oferă posibilitatea de stabilire manuală, la cerere sau automată într-un interval specificat a unor teste de date. Atunci când se vizualizează aceste date se poate alege momentul de început sau de sfârşit astfel încât datele pot fi vizualizate între intervale specificate de timp.

***Magistrala de sistem***

Pentru a coordona şi controla întrega activitate a calculatorului, microprocesorul trimite nişte mesaje, denumite *semnale,* către componentele acestuia. Din punct de vedere al tipului acestor semnale, ele se pot clasifica în semnale pentru magistrala de date (de regulă cu o dimensiune de 32 sau 64 de biți), magistrala de adrese (de asemenea, pe 32 biți sau mai mult) şi magistrala de control (formată din aproximativ 15 „linii" de control ce au rolul de a iniția sau stopa diverse activități din interiorul computerului). Una dintre liniile de control este reprezentată de ceasul de sistem, care este un oscilator de cristal de înaltă frecvență (pe placa de bază îl identificăm ca fiind un mic cilindru argintiu, situat în apropierea microprocesorului).

Prin intermediul liniilor de magistrală (pentru a controla diversele activități ale calculatorului), microprocesorul trimite semnale către componentele calculatorului, componentele trimițând, la rândul lor, un răspuns către microprocesor. În unele cazuri o astfel de acțiune este controlată de un alt dispozitiv decât microprocesorul, care poate lua controlul asupra liniilor de magistrală (în acest mod se „eliberează" şi procesorul de sarcina respectivă). Secvența semnalelor trimise prin intermediul magistralei trebuie să fie coordonate extrem de precis în timp printr-o acțiune de *sincronizare*. Dacă această operație de sincronizare este asigurată în întregime de către ceasul de sistem, atunci magistrala se numeşte sincronă.

## 2.3 Familia de procesoare x86

Primul microprocesor, 4004, pe 4 biţi, a fost realizat de firma Intel în anul 1971. Acesta a fost repede îmbunătăţit, rezultând microprocesorul 8008. În 1974 Intel realizează a doua generaţie de microprocesoare al cărei reprezentant de bază este 8080. Acesta a fost primul microprocesor de uz general având o importanţă deosebită pentru industria microcalculatoarelor.

În iunie 1978 este lansat pe piaţă microprocesorul Intel 8086 (care reprezintă a treia generaţie de microprocesoare), cu aproape 3 ani înainte de apariţia primului calculator IBM PC, bazat pe designul 8080/8085, cu o mulţime similară de registre, dar extins la 16 biţi. Acesta este momentul în care începe evoluţia familiei x86. Unitatea de interfaţă cu magistrala (Bus Interface Unit) aducea fluxul de instrucţiuni către unitatea de execuţie (Execution Unit) printr-o coadă de preîncărcare, astfel încât aducerea şi execuţia erau concurente - o formă primitivă de pipelining (instrucţiunile 8086 variau de la 1 la 4 octeţi).

Primul PC lansat pe piaţă în 1981 avea însă un procesor Intel care lucra la viteza de 4,77 Mhz. Acesta nu era însă primul produs IBM care folosea un procesor x86 - un sistem de prelucrare de texte - IBM Displaywriter, proiectat la Austin, Texas, folosise deja un microprocesor x86 la viteza de 5 Mhz. De altfel, IBM nici nu era nici prima firmă, nici singura care utiliza un microprocesor x86 într-un PC. O firmă mică, Seattle Computer Products, lansase deja un sistem realizat cu un procesor x86. Această companie a dezvoltat prima versiune a sistemului de operare PC DOS, achiziţionat de Microsoft şi denumit ulterior MS-DOS.

Pentru primul PC a fost ales microprocesorul 8088, cu performanţe scăzute, datorită necesităţii de a menţine preţul mic al calculatorului. Microprocesorul 8088 admitea o magistrală internă pe 16 biţi, însă magistrala externă nu avea decât 8 biţi. Aceasta îl făcea mai uşor de conectat la cipurile pe 8 biţi ale dispozitivelor periferice şi a permis utilizarea unei memorii de dimensiuni mai mici în construcţia calculatorului. Printre caracteristici erau 4 registre generale pe 16 biţi, ce puteau fi accesate de asemenea ca registre pe 8 biţi şi 4 registre index pe 16 biţi (incluzând pointerul de stivă). Registrele de date erau deseori folosite implicit de instrucţiuni, alocări complicate ale registrelor pentru valori temporare.

Existau de asemenea patru registre segment ce puteau fi setate din registrele index.

Registrele de segment permiteau UCP să acceseze 1 megaoctet de memorie folosind tehnica numită segmentare. Instrucțiunile programului nu puteau adresa în mod direct locațiile de memorie din spațiul de adresare, ci se folosea un proces împărțit în două etape: prima dată se încărca un registru de segment cu adresa unui bloc de 64 Kb de date sau de instrucțiuni, care putea fi plasat oriunde în memoria de 1 Mb. În continuare, orice instrucțiune a microprocesorului avea acces direct la orice dată sau instrucțiune aflată în blocul de 64 Kb. Pentru a se obține accesul în exteriorul blocului de 64 Kb, registrele de segment erau încărcate cu o nouă adresă, folosindu-se cele patru registre de segment: unul pentru accesul la date, al doilea pentru accesul la instrucțiuni, al treilea pentru accesul la stivă şi un registru special extra-segment. Schema bloc a microprocesorului 8088 este prezentată în figura 2.2.

Dacă pentru construcția lui 8088 au fost necesare 29.000 de tranzistoare într-o capsulă cu 40 de pini şi cu o tehnologie de 3 microni, microprocesoarele Pentium 4 actuale sunt construite cu 55 milioane de tranzistoare cu o tehnologie de 0,13 microni! Cu privire la evoluția numărului de tranzistori integrați pe un cip există faimoasa lege a lui Gordon Moore (cofondator la Intel) care a prezis acum 30 de ani că numărul de tranzistoare de pe un cip se va dubla la fiecare 18 luni, lege ilustrată în figura 2.3.

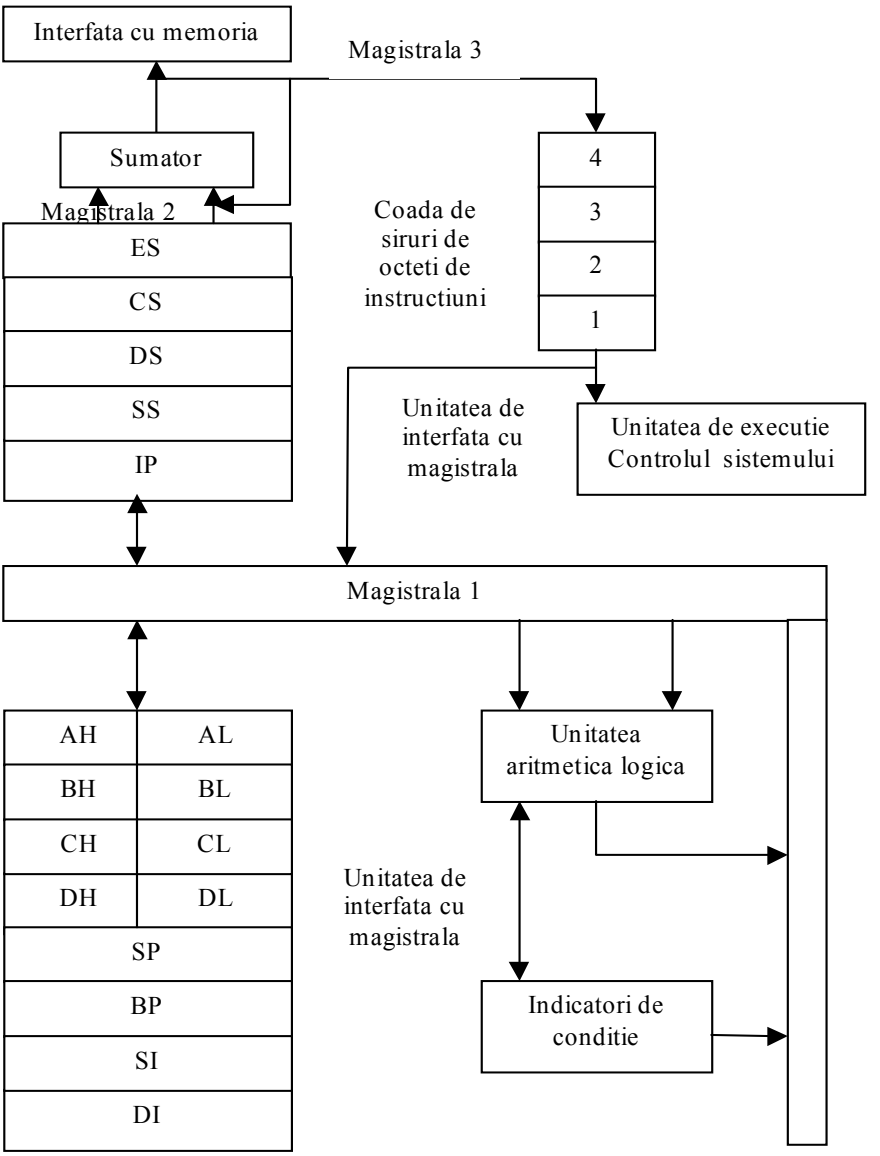



Fig. 2.2 Schema bloc a microprocesorului 8088

### *2.3.1 Microprocesorul 80286 (286)*

Familia x86 se măreşte în 1982 cu microprocesorul 80186, structural identic cu 8086 dar conţinând câteva dispozitive sistem integrate în acelaşi circuit. În acelaşi an este introdus şi 80286 care are o arhitectură extinsă faţă

de 8086. El poate lucra exact ca un 8086 având însă unele facilităţi în plus. Începând cu acest microprocesor au fost introduse mecanismele de memorie virtuală, de multitasking şi de protecţie.

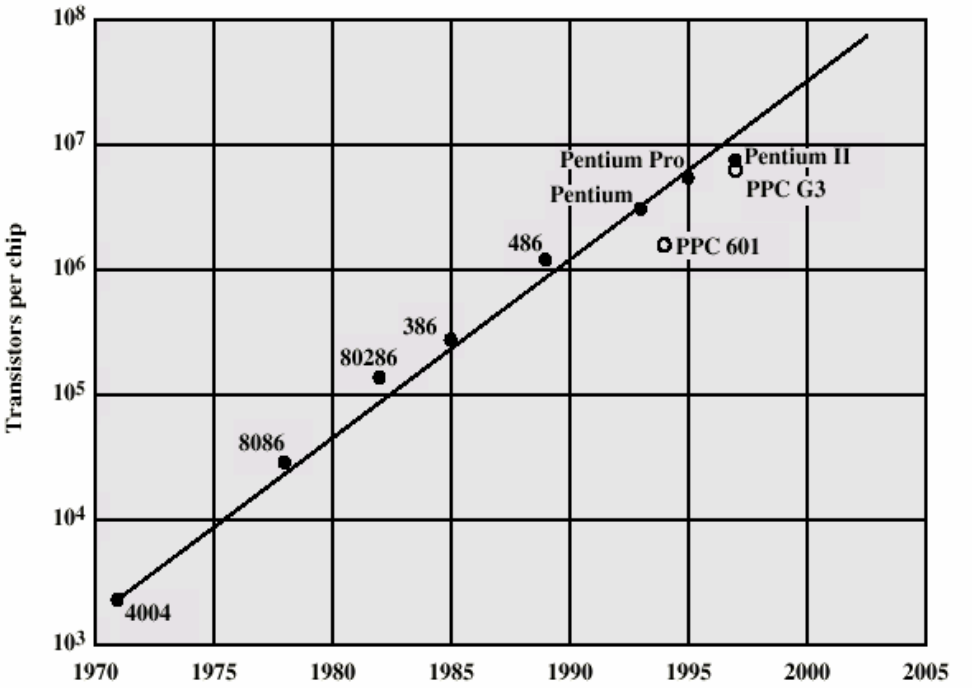



Fig. 2.3 Evoluţia numărului de tranzistoare integrate pe un cip

Calculatorul PC-AT (Advanced Tehnology - Tehnologie Avansată) 286 era construit cu 134.000 de tranzistori înglobaţi într-o capsulă cu 68 de pini. Cea mai importantă caracteristică a sa o reprezenta creştere spaţiului de adresare la 16 MB, permiţând lucrul în modul protejat şi cu memorie virtuală. Despre aceste concepte vom discuta mai pe larg în continuare. Modul protejat de lucru a permis posibilitatea folosirii sistemelor de operare multitasking, cum ar fi UNIX sau diverse variante ale acestuia.

**Memoria virtuală**

Memoria virtuală reprezintă un tip imaginar de memorie folosit de unele sisteme de operare, precum Windows (dar nu şi DOS), prin care deficitul de memorie RAM este suplinit folosindu-se hard disk-ul calculatorului. Memoria virtuală poate fi considerată ca o mulţime de adrese

de memorie; programele folosesc aceste adrese virtuale în locul celor reale pentru a stoca instrucţiuni şi date. În momentul execuţiei programului, adresele virtuale sunt convertite în adrese reale de memorie, făcându-se transferul datelor din memoria virtuală (de pe disc) în memoria reală.

Scopul folosirii memoriei virtuale este acela de a mări spaţiul de adresare, adică mulţimea de adrese de memorie pe care un program o poate utiliza. De exemplu, memoria virtuală poate conţine de două ori mai multe adrese decât memoria principală. Din această cauză, un program ce foloseşte toată memoria virtuală nu poate fi încărcat în întregime în memorie în acelaşi timp; totuşi, calculatorul poate încărca în memoria principală doar porţiunile din program necesare la un moment dat pe parcursul execuţiei. Pentru facilitarea copierii conţinutului memoriei virtuale în memoria reală, sistemul de operare împarte memoria virtuală în pagini ce conţin fiecare un număr fix de adrese. Fiecare pagină este stocată pe disc până în momentul în care este nevoie de ea.

În acel moment, sistemul de operare o copiază de pe disc în memoria reală, prin procesul de translatare a adreselor virtuale în adrese reale. Acest proces de translatare se mai numeşte şi *mapare*, iar mecanismul de copiere a paginilor virtuale de pe disc în memoria principală se numeşte *swapping* (uneori şi paginare).

### *2.3.2 Microprocesorul 80386 (386)*

Lansat pe piaţă în octombrie 1985, procesorul 80386 a apărut iniţial cu o magistrală de sistem pe 32 de biţi, apoi o versiune a acestuia cu o magistrală pe 16 biţi - denumită 386SX, în timp ce modelul pe 32 de biţi a fost redenumit 386DX. Procesorul 386SX era foarte popular, deoarece putea folosi arhitecturile 286 deja existente, cu câteva schimbări minore. Concurenţa acerbă a firmei AMD (Advanced Micro Devices), principalul producător de microprocesoare clonă a făcut ca preţul sistemelor PC complete să scadă foarte mult, înregistrându-se pentru prima dată preţuri cu mult sub 1000$. Microprocesoarele 80386DX şi 80386SX erau construite folosindu-se iniţial o tehnologie CMOS de 1,5 microni, apoi de 1 micron, conţinând 275.000 de tranzistori, în timp ce versiunea 80386SL includea 855.000 de tranzistori.

### Caracteristici ale procesorului 80386

Microprocesoarele 386 au beneficiat de câteva îmbunătăţiri semnificative faţă de microprocesoarele anterioare din familia x86. Posibilitatea prelucrării informaţiei pe 32 de biţi şi accesul la memorie printr-o magistrală de 32 de biţi au permis pentru prima dată adresarea a 4 Giga octeţi de memorie în mod direct şi a 64 Tera octeţi în mod virtual.

De asemenea, vitezele procesorului erau limitate de performanţele memoriei sistemului, folosindu-se în scopul "alimentării" microprocesorului cu mai multe instrucţiuni şi date metode ce erau folosite până atunci doar la calculatoare mainframe sau minicalculatoare. Metodele constau în folosirea bancurilor de memorie intercalată (memoria sistemului era împărţită în blocuri ce puteau fi accesate simultan pentru a creşte viteza de lucru) şi a memoriilor cache rapide. Se permitea, de asemenea, folosirea unui model de adresare complet liniar, fără a se utiliza registrele de segment, precum şi folosirea paginării.

Microprocesorul 386 putea fi folosit pentru a simula comportamentul mai multor procesoare 8086 ce lucrează în mod real, oferind posibilitaţi de multitasking. Ca şi în cazul microprocesorului 286, software-ul nu folosea pe deplin capacităţile acestuia, astfel încât abia după zece ani, o dată cu apariţia sistemului de operare pe 32 de biţi Windows 95 s-au putut valorifica cu adevărat capacităţile procesorului 386.

Arhitecturile de sistem bazate pe microprocesoare 386 s-au schimbat radical faţă de arhitecturile anterioare. Astfel, subsistemele memoriei s-au conectat la magistrala locală a microprocesorului iar folosirea memoriilor cache rapide şi a metodelor de intercalare a memoriei au devenit lucruri obişnuite. Magistrala AT a devenit doar o magistrală de extensie pentru conectarea adaptoarelor echipamentelor periferice, nemailucrând la frecvenţa ceasului microprocesorului. Primele cipuri rulau la 12,5 MHz şi apoi transformate rapid pentru a rula la frecvenţe de 16, 20, 25 ,33, 40 şi 50 de MHz.

### Microprocesorul 80386SL

Versiunea 386SL lansată de Intel era un procesor 386 special ce îngloba 855.000 de tranzistori (de 3,1 ori mai mult decât la un 386 obişnuit), fiind proiectat pentru piaţa calculatoarelor notebook, aflată în creştere. Acest microprocesor dispunea de un sistem de gestionare a consumului (Power Management), de o memorie cache de 8K, de un controeller DRAM şi era capabil să funcţioneze la 3,3 volţi. Produsul avea un concurent puternic

reprezentat de versiunea 386SX a firmei AMD, înregistrând un succes limitat, în cele din urmă fiind retras de pe piață de firma Intel.

### *2.3.3 Microprocesorul 80486 (486)*

Firma Intel a prezentat în aprilie 1989 procesorul 486, ce dispunea de puține îmbunătățiri față de 386. Setul de registre, caracteristicile de gestiune a memoriei şi tipurile de date erau aproape neschimbate față de cele de la 386, în schimb performanţele erau crescute semnificativ prin integrarea pe cip a coprocesorului matematic şi a unei memorii cache de nivelul 1 de 8K. Tehnologia de fabricație era de tipul CMOS, înglobând 1,2 milioane de tranzistori pe cip. Prima versiune, 486DX folosea o tehnologie de fabricație de 1 micron pentru frecvențele de ceas de 25 şi 33 Mhz, scăzând apoi la 0,8 microni la 486DX-50.

Micşorarea dimensiunilor tranzistorilor a prezentat două avantaje deosebite de care au beneficiat din plin microprocesoarele 486: în primul rând, fiind mai mici, folosesc mai puțină energie, iar în al doilea rând, pot lucra mai rapid. Alte dimensiuni tehnologice de fabricație erau: pentru 486DX2 - 0,8 microni (Intel), 0,5 microni (AMD), 0.65 microni (Cyrix), iar pentru 486DX4 - 0,6 microni (Intel - cu 1,6 milioane de tranzistori), 0,5 microni (AMD), 0.65 microni (Cyrix).

**Magistrala locală a microprocesorului 486**

Această magistrală s-a modificat sensibil față de cea a microprocesorului 386, permițând o creştere a ratelor de transfer pe magistrală. În timp ce magistrala veche 386 trimitea o adresă pentru fiecare acces la memorie, magistrala 486 transmitea o adresă pentru un bloc de până la 16 octeți. S-a permis astfel creşterea cu 50% a ratei de transfer față de rata de transfer de la 386, la aceeaşi frecvență şi dimensiune a magistralei. Făcându-se transferuri de blocuri de date, se trimite doar adresa blocului inițial către memorie, această tehnică fiind ideală pentru sisteme cu memorie cache, cum este şi cel 486. Transferul pe blocuri de date se face la citire cât şi la scriere şi sunt necesare două impulsuri de ceas pentru primul transfer pe 32 de biți şi câte un impuls de ceas pentru transferurile următoare pe 32 de biți. Numărul de transferuri ulteroare pe 32 de biți este de trei, astfel că pentru un procesor la 50 MHz, transferul unui bloc de maxim 16 octeți se

face pe perioada a cinci perioade de ceas, adică 160 M/secundă (față de o rată maximă de transfer pe magistrală de 1M/secundă la primul PC !).

Deoarece diferența de preț dintre procesorul 486 şi clonele 386 rapide erau foarte mari, firma Intel a hotărât să scoată pe piață o versiune mai ieftină a procesorului 486, mai puțin performantă, denumită 486SX. Diferența esențială față de microprocesorul 486 obişnuit era lipsa funcțiilor oferite de coprocesorul matematic, iar prețul era de până la o treime față de acesta, denumit ulterior 486DX. Trebuie făcută o precizare cu privire la titulatura impusă de Intel: în timp ce pentru familia procesoarelor 386, SX reprezenta un procesor pe 16 biți, pentru familia procesoarelor 486, SX reprezenta lipsa coprocesorului matematic!

Ca şi în cazul microprocesorului 386SL, Intel a lansat pe piață şi o versiune 486 pentru calculatoarele notebook, denumită 486SL, oferind caracteristici avansate de gestiune a consumului şi posibilitatea de a lucra la 3,3 volți; succesul a fost limitat şi modelul a fost retras de pe piață în scurt timp.

### *2.3.4 Microprocesorul Pentium*

În anul 1993 şi-a făcut apariția procesorul Pentium al firmei Intel, având o arhitectură superscalară (se puteau executa în anumite condiții două instrucțiuni simultan în acelaşi ciclu de ceas) şi memorii cache de nivelul unu separate pentru instrucțiuni şi pentru date, de câte 8K. O altă îmbunătățire o reprezenta creşterea dimensiunii magistralei locale Pentium la 64 de biți, iar a vitezei de lucru la 60 sau 66 de MHz.

Tehnologia de fabricație era CMOS bipolară, de 0,8 microni (pentru primele Pentium-uri la 60 şi 66 MHz), de 0,6 microni (la modelele ce lucrau la frecvențe de 75, 90, 100 MHz) şi de 0,35 microni (la modelele cu frecvențe de 120, 133, 150, 166, 200 MHz) şi înglobau de la 3,1 milioane de tranzistoare (La primul Pentium) până la 3,3 milioane la modelele ulterioare.

Referitor la denumire, se impune specificarea că acesta a fost primul microprocesor cu aceeaşi arhitectură x86, care a renunțat la numele de x86 (ar fi trebuit să se denumească 586), deoarece firma Intel nu a putut să oprească alte firme producătoare de microprocesoare clonă să folosească acest nume.

**Microprocesorul Pentium Pro ("P6")**

Microprocesorul Pentium Pro a fost introdus în 1995 ca succesor al procesorului Pentium, aducând câteva caracteristici nemaiîntâlnite până atunci într-un microprocesor de PC. Microprocesorul Pentium Pro a fost primul microprocesor ce a schimbat radical modul de execuţie al instrucţiunilor, translatându-le în microinstrucţiuni gen RISC şi executându-le apoi mai uşor în unităţile interne.

Disponibil în multiple variante cu frecvenţe între 150 MHz (noiembrie 1995) şi 200 de MHz (ianuarie 1997), microprocesoarele Pentium Pro sunt construite folosindu-se o tehnologie de 0,6 până la 0,35 microni pentru microprocesor şi memoria cache integrată de nivel 2. Numărul de tranzistori folosiţi este de aproximativ 5,5 milioane, iar dimensiunea memoriei cache de nivel 2 variază de la 256 KB la 1MB pentru microprocesoarele ce rulează la fecvenţe de 200 de MHz. Cu toate că este vechi de mai bine de trei ani, procesorul Pentium Pro este încă folosit în sisteme high-end (servere, în special), cu toate că Pentium II (şi mai nou, Pentium III) începe să-i ia locul în unele cazuri. Pentru astfel de servere ce dispun de mai multe procesoare Pentium Pro, raportul preţ/performanţă este destul de bun, concurând în acest sens chiar cu unele procesoare (non Intel) pentru servere ale firmei DEC (Digital Equipment Corporation) Alpha.

Principalele caracteristici ale procesorului Pentium Pro sunt următoarele:

- *superpipelining* - arhitectură în conductă superioară celei de la Pentium, mărindu-se numărul de stadii în execuţie de la 5 la 12;
- *memorie cache de nivelul 2 integrată pe cip* - în loc să folosească o memorie cache integrată pe placa de bază ce funcţionează la viteza magistralei de memorie, se foloseşte o memorie cache integrată în cip, având propria magistrală şi rulând la întreaga viteză a microprocesorului, în general de trei ori mai rapid decât la procesoarele Pentium. De asemenea, arhitectura memoriei cache este diferită, de tip non-blocking, ceea ce înseamnă că atunci când avem de-a face cu un "cache miss" (greşeală de memorie cache), procesorul îşi continuă execuţia, neaşteptând (re)citirea datelor din memoria principală;
- *optimizare pentru 32 de biţi:* Pentium Pro este optimizat pentru a rula coduri pe 32 de biţi (folosite de majoritatea sistemelor de operare şi a aplicaţiilor), ducând la mai bune performanţe faţă de Pentium atunci când se foloseşte software de ultimul tip;

- *mai mare adresabilitate a magistralei:* datorită faptului că magistrala de adrese este pe 36 de biți, rezultă o dimensiune maximă a adreselor de memorie fizică de 64 GB;
- *multiprocesare superioară:* există configurații de multiprocesare cu până la patru procesoare Pentium Pro (sisteme quad), față de maximum două la Pentium (sisteme duale);
- *încheierea instrucțiunilor "out of order" (neordonate)*, ceea ce înseamnă că instrucțiunile din stadiile în conductă se pot executa în altă ordine decât cea stabilită inițial;
- *previziune superioară a ramificărilor:* dimensiunea BTB (Branch Target Buffer - Memorie Buffer pentru Ramificări) este dublă față de Pentium şi acuratețea este crescută (BTB este o mică memorie asociativă (de regulă cu 128 până la 512 de intrări) ce urmăreşte indexul memoriei Icache şi încercă să prezică ce index Icache va fi accesat în continuare, bazându-se pe istoria ramificărilor. Optimizarea actualului algoritm folosit în acest scop reprezintă un domeniu continuu de cercetare, procesorul Pentium Pro folosind o variantă a algoritmului lui Yeh - 1991);
- *redenumirea registrelor*: această faciliate oferă creşterea performanțelor paralele ale conductelor;
- *execuție speculativă:* un mecanism generalizat ce permite instrucțiunilor să fie prelucrate "mai devreme", înainte de ordinea de execuție normală. Rezultatele acestor speculații (previziuni) sunt stocate temporar (în ROB - Re-Order Buffer - Buffer de Reordonare - care este o unitate funcțională a procesorului Pentium Pro unde micro instrucțiunile inițiale aşteaptă, rezultatele speculative sunt colectate şi apoi retrase) deoarece există posibiliatea să nu fie utilizate datorită unor schimbări în execuția programului.
- *tehnologie de execuție dinamică* care poate fi concepută pe scurt ca o ajustare optimală a execuției instrucțiunilor prin prevederea desfăşurării programului, analizând graficul desfăşurător al programului pentru a alege cea mai bună ordine de execuție a instrucțiunilor şi apoi având abilitatea de execuție speculativă a instrucțiunilor în ordinea preferată. Pentru a înțelege mai bine implementarea execuției dinamice, figura 2.4. ne prezintă cele trei porțiuni interne de interfață cu subsistemul de memorie.

Apar în această figură patru componente ale structurii pipeline ale procesorului Pentium Pro:

- *Unitatea FETCH/DECODE*, care este o componentă ce preia ca input fluxul de instrucțiuni al programului utilizatorului din cache-ul de instrucțiuni şi apoi le decodifică într-o serie de microoperații (numite *uops*) ce reprezintă ordinea fluxului de instrucțiuni.
- *Unitatea DISPATCH/EXECUTE* este o unitate "out-of-order" ce acceptă ordinea fluxului, programează execuția microoperațiilor în conformitate cu dependența de date, existența resurselor, stocând temporar rezultatele acestor execuții speculative.
- *Unitatea RETIRE* este componenta "in-order" ce ştie cum şi când să aplice ("retire") rezultatele speculative temporare în stări arhitecturale permanente.
- *Unitatea BUS INTERFACE* este o componentă parțial ordonată responsabilă cu conectarea celor trei unități interne cu lumea reală. Această unitatea de interfață cu magistrala comunică direct cu cache-ul L2 şi suportă până la patru accesări cache concurente.

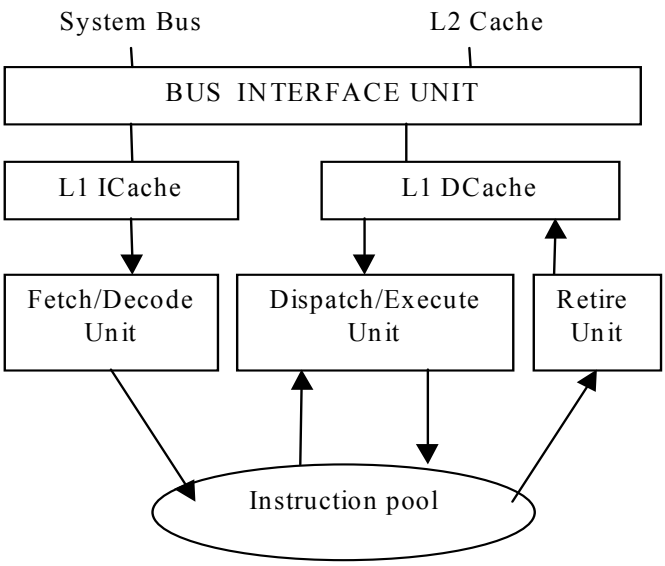



Fig. 2.4 Cele 3 unități interne de interfață cu subsistemul de memorie ce foloseşte memorii cache unificate de câte 8K

### *2.3.5 Microprocesorul Pentium MMX*

Microprocesorul Pentium MMX (varianta P55C) a fost introdus pentru prima dată în ianuarie 1997 şi este primul microprocesor Pentium ce a inclus instrucţiuni MMX, urmat de AMD K6, Cyrix 686MX şi Pentium II. Creşterea performanţelor faţă de procesorul Pentium constă în abilitatea deosebită de a opera cu aplicaţii multimedia mult mai eficient, fiind disponibil în variante ale frecvenţei de ceas de 166, 200 şi 233 MHz pentru sistemele desktop şi 133, 150, 166, 200 şi 233 MHz pentru sistemele mobile.

Îmbunătăţiri aduse microprocesorului Pentium clasic:

- Noi instrucţiuni. Inginerii firmei Intel au adăugat 57 de noi instrucţiuni proiectate special să manipuleze şi să proceseze date video, grafice şi audio mai eficient. Aceste instrucţiuni sunt orientate către secvenţe paralele, repetitive, ce se găsesc deseori în operaţii multimedia.
- SIMD (Single Instruction, Multiple Data). Aplicaţiile multimedia şi de comunicaţie de astăzi folosesc uzual cicluri repetitive ce ocupă sub 10% din codul total al aplicaţiei şi folosesc 90% din timpul de execuţie al aplicaţiei. Procedeul SIMD permite ca o instrucţiune să se execute pentru mai multe seturi de date simultan. Se reduc astfel ciclurile intens calculatorii referitoare la video, audio, grafică şi animaţie. De asemenea, mulţimea instrucţiunilor MMX are permite folosirea a două instrucţinui MMX într-un singur ciclu de ceas, deci două instrucţiuni SIMD pot fi procesate cu date pe 16 biţi într-un singur ciclu de ceas.
- Mai multă memorie cache. Firma Intel a dublat memoria cache internă la 32KB (16 KB pentru date, 16 KB pentru instrucţiuni ), astfel că se pot stoca mai multe date şi instrucţiuni pe cip, reducându-se numărul de accesări la memoria ami lentă din afara microprocesorului.
- Îmbunătăţiri bazate pe caracteristicile altor microprocesoare. Pentium MMX foloseşte o unitate de predicţie a ramificărilor, concept preluat de la Pentium Pro, având, de asemenea, o implementare a stivei de reîntoarcere ("return stack") – concept al microprocesorului IBM/Cyrix 6x86.
- Alte îmbunătăţiri. Microprocesorul Pentium MMX are patru memorii buffer de scriere faţă de două la Pentium-ul clasic, iar conductele U şi V au fost crescute cu un stadiu. A fost îmbunătăţită, de asemenea, capacitatea de procesare paralelă a celor două conducte. Figura 2.5 prezintă implementarea tehnologiei Intel MMX. Se pot observa aici cele

două conducte (MMX U pipe şi MMX V pipe) adăugate în paralel la structura internă deja existentă la Pentium.

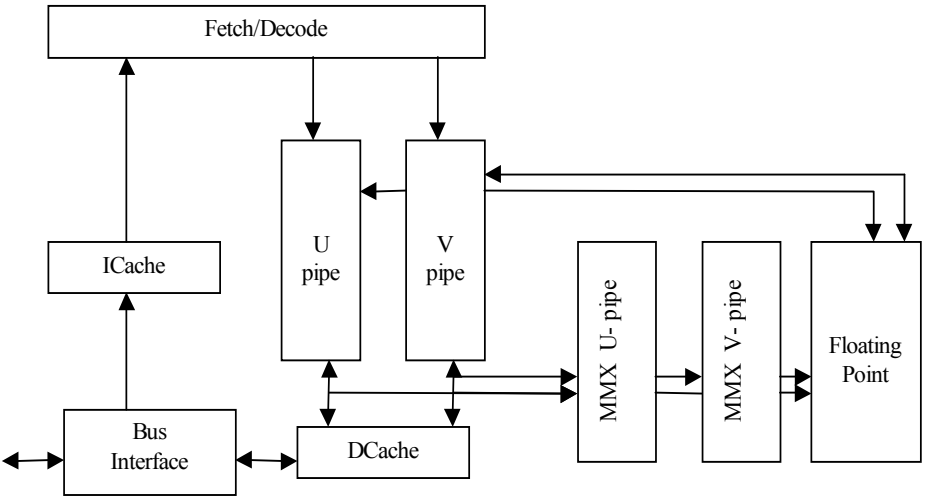



Fig. 2.5. Diagrama implementării tehnologiei Intel MMX

### *2.3.6 Microprocesorul Pentium II ("Klamath")*

Apărut în luna mai a anului 1997, Pentium II, cu numele de cod "Klamath" (după numele fluviului din Oregon, SUA), reprezentând un pas important în evoluția de la Pentium Pro. Se pare că obiectivele firmei Intel pentru crearea procesorului Pentium II au fost, în principal, să reducă costurile deosebit de mari ale memoriei cache integrate de nivel 2, care constituia principala dificultate în construcția Pentium-ului Pro. Din punct de vedere arhitectural, Pentium II nu este foarte diferit de Pentium Pro, cu o parte internă de emulare x86 şi înglobând majoritatea caracteristicilor acestuia. Diferențele sunt, în mare, următoarele:

- Memorie cache de nivel 1 dublă: mărimea memoriei cache interne a crescut de la 16 KB la 32 KB în total (16 KB instrucțiuni, 16 KB date);
- Cache pentru registele de segment: acestea sunt memorii cache speciale folosite pentru a ajuta Pentium II să proceseze codurile pe 16 biți mai eficient, ştiindu-se că Pentium Pro a fost optimizat pentru coduri pe 32 de biți, ceea ce înseamnă că nu se comportă la fel de bine pentru coduri pe 16 biți. Probabil ca răspuns la dominația sistemului de operare

Windows 95 a existat o dorință de îmbunătățire a performanțelor în acest sens.

- Buffere de scriere de mărime mai mare: mărimea bufferelor de scriere a crescut, ducând la o mică îmbunătățire a performanțelor.

Acestea sunt îmbunătățirile din punct de vedere arhitectural; îmbunătățiri reale au apărut prin creşterea frecvenţelor de ceas şi prin adăugarea extensiilor mulțimii de instrucțiuni MMX. Vitezele de ceas au variat de la 233 MHz la 266 MHz, apoi 300 MHz şi 333 MHz. Totuşi, cea mai mare noutate nu a reprezentat-o cipul în sine, ci modalitatea de prezentare a acestuia. Memoria cache integrată de la Pentium Pro ce rula la viteza de ceas a microprocesorului, constituia o îmbunătățire majoră adusă arhitecturii x86. Intel a înlocuit la Pentium II această memorie cu un circuit special ce conține procesorul şi 512 KB de memorie cache secundară ce rulează la jumătate din viteza microprocesorului. Acest ansamblu, denumit **SEC** (**S**ingle-**E**dge **C**artridge - cartuş cu o singură muchie), a fost conceput să se potrivească într-un soclu cu 242 de pini pe placa de bază Pentium II.

Această schimbare de proiectare a stârnit diverse controverse, accentuate de faptul că noua interfață Slot 1, patentată de Intel, nu va putea fi folosită de companiile rivale AMD şi Cyrix pentru noile lor procesoare.

### *2.3.7 Microprocesorul Pentium III*

Inițial, microprocesorul original Pentium III (nume de cod *Katmai*) nu era foarte diferit de Pentium II. De-abia cu noua variantă îmbunătățită de Pentium III (nume de cod *Coppermine*) Intel a reuşit să producă ceea ce vroia cu adevărat, dar lucru realizat cu întârziere. Acest microprocesor are o memorie cache L2 de 256 Kb integrați pe plăcuța de siliciu (aşa numita memorie *on-die*). În combinație cu noile chipset-uri 810/820/840 oferă suport pentru 4 porturi AGP, 133FSB şi memorie RDRAM (Rambus). Procesoarele Pentium III (nume de cod *Willamette*) au, de regulă, viteze cuprinse între 800 şi 1200 MHz.

### *2.3.8 Microprocesorul Pentium 4*

Cea mai nouă arhitectură existentă în acest moment, Pentium 4 are câteva avantaje comparativ cu celelalte procesoare x86: o latență mai mică a

memoriilor cache L1 şi L2, execuţia instrucţiunii de adunare în jumătate de tact şi execuţia unui număr de 126 de instrucţiuni de tip RISC.

Pentium 4 poate procesa mai multe instrucţiuni în paralel datorită unităţii aritmetico-logice duble şi a memoriilor tampon asociate mai mari. Planificatoarele integrate pot găsi mai multe instrucţiuni independente, având în vedere numărul de 126 de instrucţiuni ce pot fi derulate. Unităţile aritmetico-logice pot executa patru instrucţiuni într-un ciclu de tact.

Procesorul Intel® Pentium® 4 cu tehnologie HT (Hyper Threading) oferă cea mai mare putere de procesare pentru aplicaţiile avansate din zilele noastre. Tehnologia Hyper-Threading de la Intel permite procesorului să execute două fire de execuţie în paralel, astfel încât software-ul poate rula mai eficient prin implementarea mai bună a multitasking-ului. Pentium 4 dispune de o magistrală de sistem ce acţionează la 800 MHz iar viteza de ceas este cuprinsă între 2,4 şi 3,2 GHz. Micro-arhitectura de la Intel se numeşte NetBurst®, bazată pe tehnologie de 0,13 microni. Procesorul Pentium 4 oferă performanţe superioare pentru procesare de imagini digitale, video, muzică digitală, jocuri 3D sau procesare DVD şi format video MPEG4.

## 2.4 Coprocesoare

### *2.4.1 Coprocesoare matematice*

În general, un coprocesor este o unitate de procesare ce asistă CPU în calculul diferitelor tipuri de operaţii. De exemplu, un coprocesor matematic poate realiza calcule matematice, în particular operaţii în virgulă mobilă. Coprocesoarele matematice se mai numesc coprocesoare numerice sau coprocesoare în virgulă mobilă.

Istoria microprocesoarelor din familia x86 este strâns legată de cea a coprocesoarelor matematice. Coprocesorul matematic a fost denumit în mai multe moduri: pur şi simplu coprocesor, coprocesor matematic, procesor de virgulă mobilă, *NPX (Numerical Processor eXtension)*, această denumire din urmă fiind cea acceptată (şi preferată) de firma Intel. Acesta este un cip special ce poate efectua operaţii în virgulă mobilă, calcula funcţii matematice - trigonometrice, logaritmice, ş.a.m.d - ce a primit codul x87 (8087). De remarcat este faptul că, pentru a benficia de avantajele unui

coprocesor trebuie ca programul respectiv să conțină instrucțiuni ale coprocesorului, altfel coprocesorul nu va fi niciodată utilizat.

Arhitecturile inițiale ale primelor calculatoare personale aveau un soclu liber unde se putea monta un coprocesor matematic sistemului. Fiecărui microprocesor din această familie i-a fost asociat un coprocesor pentru a îmbunătăți performanțele sistemului de calcul. Pentru microprocesoarele 8086, 80286 80386 şi 80486SX erau folosite coprocesoarele 8087, 80287 80387 şi respectiv, 80487. Începând cu microprocesorul 486, mai precis versiunea 486DX, acest coprocesor matematic este încorporat direct în cipul microprocesorului.

Dar să explicăm mai detaliat rolul coprocesorului matematic în îmbunătățirea performanțelor sistemului de calcul. În esență, procesorul principal din familia x86 lucrează doar cu numere întregi, cu sau fără semn. În unele calcule însă, pentru a se executa rapid operații matematice, de exemplu, trebuie să se folosească numere cu un alt format decât cel întreg. Cu toate că există software specializat ce emulează lucrul cu numere în alt format decât cel întreg, această soluție este lentă şi consumă o mare parte din puterea microprocesorului sistemului. Apare astfel necesitatea folosirii coprocesorului matematic, pentru a spori viteza cu care se execută aceste calcule cu numere în format diferit de cel întreg.

Designerii procesoarelor x86 au elaborat o metodă de atașare a coprocesorului matematic la microprocesorul principal, printr-o interfață cu un nivel înalt de generalizare ce admite toate tipurile de coprocesoare specializate, folosită în special pentru coprocesorul NPE x87. Astfel introduse, coprocesoarele matematice sunt capabile să efectueze operațiile numerice de 20-100 de ori mai rapid decât software-ul specializat în emularea acestor operații. Aplicațiile ce includ calculul tabelar, aplicațiile ştiințifice sau tehnice, aplicațiile multimedia sau cele de proiectare asistată de calculator *(CAD - Computer Aided Design)* şi unele jocuri (Quake, spre exemplu) necesită operații numerice la viteze foarte mari şi, de aceea, beneficiază din plin de facilitățile de accelerare a funcțiilor matematice acordate de introducerea coprocesoarelor matematice.

După cum am mai spus, începând cu microprocesoarele 486DX, coprocesorul este integrat pe acelaşi cip cu acesta, în timp ce până atunci calculatoarele personale erau livrate fără coprocesorul matematic instalat, existând posibilitatea atașării sale înr-un soclu liber de pe placa de bază. Trebuie menționat, de asemenea, că performanțele matematice în virgulă mobilă ale PC-urilor bazate pe microprocesoare Intel din familia x86 (având coprocesor matematic ataşat sau integrat în procesorul principal) erau mult

inferioare celor obținute de calculatoarele personale bazate pe procesoare RISC. De aceea, firma Intel a căutat să le îmbunătățească, astfel că a reproiectat în întregime coprocesorul matematic al microprocesorului Pentium, obținând performanțe de calcul în virgulă mobilă de până la 10 ori mai mari decât la microprocesorul 486, apropiindu-se în acelaşi timp de performanțele coprocesoarelor RISC.

Alături de coprocesoarele matematice, există, de asemenea, şi coprocesoare grafice (microprocesoare proiectate special pentru a realiza calcule grafice) pentru manipularea imaginilor grafice. Acestea sunt adesea numite acceleratoare grafice (accelerator boards).

### *2.4.2 Tipuri de date admise de către un coprocesor matematic*

Un coprocesor matematic poate lucra cu numere în virgulă mobilă, precum şi cu date reprezentate în mod întreg sau cod BCD (Binary Coded Decimal – cod în care două cifre zecimale se reprezintă pe un octet). În tabelul 2.6. sunt evidențiate tipurile de date numerice, cifrele semnificative şi domeniile de definiție admise de coprocesoarele Intel. De asemenea, în tabelul 2.7. sunt ilustrate principalele instrucțiuni ale coprocesorului matematic, funcțiile algebrice şi transcendente, constantele aritmetice incluse în coprocesor şi folosite deseori în calcule din matematica superioară.

Calculul datelor numerice în interiorul coprocesorului sunt făcute prin translatarea acestora într-o reprezentare internă standard numită reprezentare temporară reală (pe 80 de biți). Coprocesorul are în componență un set suplimentar de registre, ce constă din opt registre de date reprezentate în mod real pe 80 de biți, care se pot comporta ca o stivă de date sau care pot fi accesate în mod independent. Cu ajutorul instrucțiunilor coprocesorului se pot realiza calculele şi modificările din aceste opt registre de date.

Tabelul 2.6

**Tipurile de date cu care lucrează un coprocesor matematic, numărul de cifre semnificative şi domeniul aproximativ de definiţie**

| Tipul de date | Numărul de biţi | Cifre semnificative | Domeniul aproximativ de definiţie |
|---|---|---|---|
| Întreg tip cuvânt | 16 | 4 | -32768 ÷ 32767 |
| Întreg scurt | 32 | 9 | $-2\times10^{9} \div 2\times10^{9}$ |
| Întreg lung | 64 | 18 | $-9\times10^{18} \div 9\times10^{18}$ |
| Zecimal compact | 80 | 18 | -99...99 ÷ 99...99 (18 cifre) |
| Real simplu | 32 | 7 | $1.18\times10^{-38} < \|x\| < 3.4\times10^{38}$ |
| Real dublu | 64 | 15-16 | $2.23\times10^{-308} < \|x\| < 1.79\times10^{308}$ |
| Real extins | 80 | 19 | $3.37\times10^{-4932} < \|x\| < 1.18\times10^{4932}$ |

Tabelul 2.7

**Instrucţiunile coprocesorului matematic**

| Clasa de instrucţiuni | Tipul de instrucţiuni |
|---|---|
| Transferul datelor | Încărcare, stocare, schimb |
| Aritmetice | Adunare, scădere, înmulţire, împărţire, scădere/împărţire inversă, ridicare la putere, radical, calculul restului, calculul părţii întregi, schimbare semn, etc. |
| Comparări | Comparaţie, examinare, testare |
| Constante | $0,1,\pi,\log_{10}2,\ln2,\log_2 10,\log_2 e$ |
| Controlul asupra procesorului | Încărcare/stocare cuvânt de control, stare cuvânt de control, încărcare/stocare mediu, salvare, reîncârcare, iniţializare, anulare excepţie |
| Funcţii | Tangentă, arctangentă, sinus, cosinus, $2^x-1$, $y*\log_2 x$, $y*\log_2(x+1)$ |

Cooperarea dintre microprocesorul principal şi coprocesor se face în modul următor: pentru activarea coprocesorului matematic, microprocesorul îi transmite acestuia un cod special, numit *escape*. În acest moment se iniţiază cooperarea dintre cele două procesoare. După ce datele trimise coprocesorului au ajuns la acesta, ele încep să fie prelucrate prin funcţiile specifice coprocesorului, iar în acelaşi timp microprocesorul principal pregăteşte configurarea datelor pentru următoarea sesiune de lucru a coprocesorului. Aceste două operaţii se execută simultan, dar sunt distincte în acelaşi timp.

### *2.4.3 Procesoare DSP (Digital Signal Processor)*

Procesoarele de semnale digitale reprezintă unul dintre cele mai importante tipuri de coprocesoare existente într-un calculator personal. Acest tip de coprocesor este capabil să prelucreze semnale analogice din mediul real (imagini, sunete, etc.) ce au fost convertite în semnale digitale, conţinând în acest sens instrucţiuni folosite în manipularea semnalelor analogice şi prelucrarea acestora. Astfel, coprocesoarele de tip DSP sunt utilizate în aplicaţii de tipul:

- sinteza şi recunoaşterea vorbirii;
- prelucrarea imaginilor video şi fotografice;
- sinteza sunetelor şi a muzicii;
- comprimarea şi decomprimarea audio-video;
- accelerarea grafică 2D şi 3D;
- funcţii de modem.

Au apărut astfel şi apar în continuare noi tipuri de interfeţe ce permit introducerea în sistem a unor coprocesoare DSP. Un exemplu de astfel de interfaţă este interfaţa *RMI (Resource Manager Interface - interfaţă de administrare a resurselor)* ce permite transferul unor aplicaţii de la microprocesorul principal la un coprocesor DSP care le poate executa în timpul în care microprocesorul este ocupat cu alte operaţii. Există câteva companii furnizoare de plăci de extensie dotate cu coprocesoare DSP, în special pentru aplicaţii multimedia, însă preţul acestora este încă destul de mare.

## 2.5 Extensii MMX (MultiMedia eXtension sau Matrix Math eXtension)

Necesitatea creşterii tot mai mari a performanţelor micro-procesoarelor calculatoarelor personale a dus la ambiţia firmei Intel de înglobare în funcţiunile procesoarelor Pentium a unor funcţii de prelucrare a semnalelor ce erau îndeplinite până acum de coprocesoarele DSP. Au apărut astfel instrucţiunile MMX ce au reprezentat prima schimbare majoră în setul de instrucţiuni x86 în ultimul deceniu. MMX defineşte 57 de noi instrucţiuni şi un prim gol de compatibilitate de la introducerea microprocesorului 386. Toate noile procesoare ale firmelor Intel, AMD sau Cyrix înglobează suport pentru aceste noi instrucţiuni.

Scopul extensiei instrucţiunilor MMX este ecela de a asigura o performanţă hardware pentru anumite tipuri de programare multimedia. În particular, instrucţiunile MMX sunt proiectate să asigure procesarea unei mari cantităţi de date într-un timp scurt, folosind o singură instrucţiune. Acest procedeu este deseori referit ca *SIMD (Single Instruction Multiple Data),* adică execuţia unei singure instrucţiuni pe mai multe seturi diferite de date în acelaşi timp. Aceasta este de fapt, originea numelui *Matrix Math eXtension (Extensie Matricială Matematică),* din moment ce aceste instrucţiuni operează asupra unei matrice de date (date multiple). Aceste tipuri de instrucţiuni sunt în mod special folositoare pentru calcule necesare în procesări grafice, video şi audio, ceea ce a condus la cea de-a doua proveniență a numelui: *MultiMedia eXtensions (Extensii MultiMedia).*

Din moment ce MMX este o extensie a unui set de instrucţiuni, aceasta înseamnă că procesoarele ce vor suporta această extensie vor rula toate aplicaţiile software ce rulează pe procesoarele anterioare, dar nu şi vice-versa: procesoarele MMX sunt compatibile cu cele mai vechi, dar nu şi invers. De asemenea, pentru a utiliza avantajul instrucţiunilor MMX, trebuie ca şi software-ul respectiv să fie scris având în vedere aceste extensii. În acest sens, software-ul trebuie să detecteze dacă procesorul suportă MMX, deoarece procesoarele non-MMX nu pot procesa deloc instrucţiuni MMX, apoi este posibil ca software-ul să fie scris astfel încât să lucreze atât cu instrucţiuni MMX cât şi cu instrucţiuni non-MMX, doar folosind coduri diferite în zonele în care sunt folosite instrucţiunile MMX.

În mod evident, deoarece se referă în mod special la aplicaţiile multimedia, celelalte aplicaţii vor beneficia foarte puţin de avantajele extensiei MMX. Procesările video, editarea fişierelor audio, prelucrările grafice, jocurile şi software-ul similar cu acestea beneficiază în mod sigur de performanţele crescute sub extensia MMX, în timp ce aplicaţiile standard

nu au decât foarte puțin de câştigat sau deloc de pe urma avantajelor tehnologiei MMX.

## 2.6 Elemente de arhitectură internă a procesorului

### *2.6.1 Registrele*

Registrele microprocesorului reprezintă suprafețe locale de memorare (locații de memorie) în interiorul microprocesorului, folosite pentru stocarea datelor cu care se lucrează. Fiecare procesor are un anumit număr de registre, unele dintre acestea dedicate unor funcțiuni speciale iar altele disponibile pentru uzul general al programatorilor. Registrele reprezintă memoria cea mai rapidă existentă într-un PC, chiar mai rapidă decât memoria cache de nivelul 1, deoarece ele sunt integrate direct în logica microprocesorului.

Cele mai multe operații se fac asupra registrelor; procesorul nu poate executa o operație aritmetică, spre exemplu, direct în memorie. Dacă dorim să adunăm valoarea 1 unei locații de memorie, procesorul va face acest lucru în mod normal încărcând valoarea inițială din memorie într-un registru, va adăuga 1 în registru şi apoi va salva noua valoare obținută înapoi în memorie. Această complicată "schemă" se petrece, evident, foarte repede şi transparent față de programator.

Lățimea în biți a registrelor microprocesorului determină cantitatea de date cu care se pot efectua calcule la un moment dat. Această mărime este folosită uneori pentru a eticheta "mărimea" procesorului. De exemplu, putem auzi pe cineva vorbind despre un "procesor pe 16 biți", "procesor pe 32 de biți" sau de un "procesor pe 64 de biți". Acest termen se referă în mod normal la mărimea registrelor din interiorul CPU, însă este deseori greşit folosit, atribuirea "mărimii" microprocesorului făcându-se prin referire la lățimea magistralei de date. În acest sens este interesant faptul că toate microprocesoarele începând cu 386 şi până la Pentium III sau K7 sunt microprocesoare pe 32 de biți, bazându-ne pe această definiție. Doar ultimile microprocesoare RISC, cu ar fi Alpha-DEC, precum şi viitorul Itanium (bazat pe noua arhitectură IA-64 şi EPIC*) de la Intel sunt microprocesoare pe 64 de biți. De asemenea, ultima generație de procesoare

---

* *EPIC (Explicitly Parallel Instruction Computing)* reprezintă conceptul ce are ca prim obiectiv reducerea complexității UCP prin optimizarea compilatoarelor.

AMD, Athlon 64 3400+ este un procesor pe 64 de biți aflat în competiție cu Pentium 4.

Cu cât numărul de registre este mai mare într-un microprocesor, programatorii au o mai mare flexibilitate în scrierea unui mai bun cod-maşină. Totuşi, aceasta implică creşterea complexității procesorului, pe de o parte, dar şi necesitatea compatibilității cu modelele anterioare de microprocesoare, pe de altă parte.

### *2.6.2 Unitatea de interfață cu magistrala*

Unitatea de interfață cu magistrala *(Bus Interface Unit)* este partea microprocesorului ce asigură interfața cu restul calculatorului. Numele său vine de la faptul că se ocupă cu manipularea informației prin magistrala de date a microprocesorului, condiție primordială pentru transferul datelor de la şi către CPU. Unitatea de interfață cu magistrala este răspunzătoare pentru toate semnalele ce se transmit către procesor, precum şi pentru generarea tuturor semnalelor ce pleacă de la procesor către alte componente ale sistemului de calcul.

### *2.6.3 Unitatea de control*

Unitatea de control *(Control Unit)* reprezintă circuitele ce controlează fluxul de informații prin procesor şi coordonează activitățile celorlalte componente ale sale. Această componentă poate fi denumită, într-o oarecare măsură, "creierul din interiorul creierului", din moment ce controlează ce se întâmplă în interiorul procesorului, iar acesta din urmă controlează restul calculatorului.

Funcțiunile executate de către unitatea de control variază în mare măsură în funcție de arhitectura internă a CPU, din moment ce unitatea de control implementează efectiv această arhitectură. Pentru un procesor obişnuit ce execută (nativ) instrucțiuni x86, unitatea de control îndeplineşte funcțiunile de aducerea a instrucțiunilor din memorie, de decodificare a acestora, de control al execuției şi de stocare în memorie.

Pentru un microprocesor RISC, unitatea de control are o serie întreagă de funcțiuni în plus. În acest sens, gestionează translatarea instrucțiunilor x86 în microinstrucțiuni RISC, coordonează planificarea microinstrucțiunilor între diferite unități de execuție şi asigură ca rezultatele obținute de la aceste unități de execuție diferite să ajungă acolo unde trebuie. Pentru unele astfel de microprocesoare unitatea de control se poate

împărți în alte subunități de control (precum unitatea de planificare ce asigură planificarea şi o unitate de aşteptare – (*retirement unit)* ce asigură controlul rezultatelor ce vin din pipeline) datorită complexității deosebite a activităților ce trebuiesc efectuate şi gestionate.

### *2.6.4 Unitatea de execuție a întregilor*

Unitatea de execuție a întregilor *(Integer Execution Unit)* se ocupă cu operații cu numere întregi. Majoritatea calculelor făcute în interiorul PC-ului sunt făcute cu date în format întreg. Denumirea de "întreg" include numere întregi, caractere (text) sau altfel de date similare. Numerele care nu sunt întregi se numesc numere "în virgulă mobilă" (*floating point numbers*). Acestea sunt tratate diferit, folosindu-se o unitate special dedicată, numită unitate de virgulă mobilă (FPU - Floating Point Unit). Există unitați de întregi care pot manipula operații cu numere în virgulă mobilă, dar foarte încet în comparație cu unitățile în virgulă mobilă dedicate.

Unitatea de execuție a întregilor este locul în care instrucțiunile sunt executate şi se fac calculele. Procesoarele mai vechi aveau o singură astfel de unitate, iar instrucțiunile erau procesate secvențial. Modelele mai noi folosesc mai multe unități de execuție, permițând în acest fel ca instrucțiunile să fie executate simultan, crescând performanțele. Procesoarele construite pe baza acestei concepții se spune că au o *arhitectură superscalară*. Procesoarele şi mai avansate pot dispune de unități de execuție proiectate special pentru execuția unor tipuri de instrucțiuni, acest lucru fiind în special valabil la procesoarele ce folosesc emularea x86 cu un nucleu procesor de tip RISC.

### *2.6.5 Unitatea de prelucrare în virgulă mobilă*

Unitatea de prelucrare în virgulă mobilă *(FPU - Floating Point Unit)* este o unitate de execuție dedicată efectuării de calcule în virgulă mobilă precum şi calculelor de funcții matematice. FPU este integrată în toate microprocesoarele începând de la 386DX încoace (mai puțin procesorul 486SX). Primele calculatoare trebuiau să folosească unitatea de execuție a întregilor pentru a desfăşura operații în virgulă mobilă (dar foarte lent), aceasta în cazul în care nu aveau ataşate un coprocesor matematic (vezi subcapitolul referitor la coprocesoare). Coprocesorul matematic lucra în colaborare cu procesorul principal pentru a îmbunătăți performanțele în cazul rulării aplicațiilor intensiv orientate pe calcule matematice: calcul tabelar, aplicații ştiințifice, multimedia, jocuri etc.).

### *2.6.6 Memoria cache de nivel 1 şi controller-ul de memorie cache*

Toate microprocesoarele moderne încorporează o mică memorie de viteză foarte mare aflată direct pe cip, numită memorie cache de nivel 1. Această memorie este folosită pentru stocarea datelor şi instrucţiunilor recent folosite sau ce urmează a fi utilizate imediat. Un principiu al ştiinţei calculatoarelor spune că dacă un procesor a referit recent o locaţie de memorie, este foarte probabil ca să facă referire din nou la acea locaţie în viitorul apropiat. Folosind o memorie ultrarapidă cache pentru a stoca datele recent folosite absolvă procesorul de căutarea în memorie a acelor date şi de a le încărca din nou. Acest lucru are o importanţă deosebită şi îmbunătăţeşte sensibil performanţele, deoarece memoria principală este cu multe ordine de mărime mai lentă decât memoria de tip cache a microprocesorului. În mod evident însă, o creştere a performanţei impune şi un preţ de plătit: memoriile cache sunt memorii foarte scumpe.

Memoria cache integrată în microprocesor se numeşte memorie cache de nivel 1 (sau memorie cache primară) deoarece este cea mai apropiată de acesta. De fiecare dată când procesorul cere informaţii ce se găsesc în memorie, controller-ul de memorie cache (sau, pe scurt, controllerul cache) de pe cip foloseşte un circuit special prin care în primă fază verifică dacă datele respective sunt deja în memoria cache. Dacă da, atunci sistemul a economisit timpul respectiv de acces la memoria principală. Cele mai multe PC-uri folosesc şi o memorie cache de nivel 2 (sau memorie cache secundară, plasată pe placa de bază a PC-ului) pentru stocarea datelor recent folosite ce nu au mai încăput în memoria cache de nivel 1 (mult mai mică).

Memoriile cache primare tipice ale microprocesoarelor variază de la 8KB (la 486) la 32KB (Pentium II) sau la 64KB (AMD K6), putând avea însă dimensiuni chiar mai mari la microprocesoarele recente de tip RISC, în timp ce microprocesoarele dinainte de 486 nu aveau memorie cache primară. Aceste memorii cache sunt foarte rapide deoarece rulează la întrega viteză a procesorului şi sunt integrate direct în acesta.

Există două modalităţi de organizare a memoriei cache primare de către un procesor; unele procesoare posedă o singură memorie cache pentru a folosi atât date cât şi instrucţiuni; o astfel de memorie cache se numeşte, de obicei, memorie cache unificată. Alte procesoare folosesc două memorii cache separate: una pentru date şi alta pentru instrucţiuni, iar modalităţile de

scriere pot fi diferite în cele două memorii. De exemplu, la microprocesorul Pentium memoria cache pentru date adoptă o politică de tipul "write-back", în timp ce pentru memoria cache de instrucţiuni este folosită doar metoda "write-through". Politica de scriere a memoriei cache determină modalitatea de scriere în locaţiile de memorie ce sunt stocate în memoria cache; din acest punct de vedere există două tipuri de memorii cache:

- Memorii cache "write-back" (sau *cache "copy back"*). Acest tip de memorie cache funcţionează astfel: când sistemul scrie într-o locaţie de memorie ce este în prezent stocată în cache, noua informaţie este scrisă doar în cache, nefiind scrisă efectiv în memoria sistemului. În continuare, dacă altă locaţie de memorie trebuie să folosească zona de cache unde informaţia este stocată, este salvată înapoi *(write-back)* în memoria principală şi apoi zona eliberată de cache poate fi folosită de noua adresă. Acest tip de cache oferă performanţe mai bune decât memoria "write-through", deoarece economiseşte timpul de scriere în memoria principală.

- Memorii cache "write-through". În acest caz, de fiecare dată când procesorul scrie într-o locaţie de memorie cache, atât memoria cache cât şi memoria sistemului sunt reactualizate, având loc deci o pierdere de timp pentru scrierea în memoria de sistem. Acest tip de cache are performanţe mai slabe decât "write-back", dar este mai simplu de implementat.

## 2.7 Elemente de arhitectură externă a microprocesorului

### *2.7.1 Procesorul şi magistralele de memorie*

Modul în care microprocesorul comunică cu celelealte părţi ale sistemului este de multe ori un factor de determinare a puterii sistemului precum şi a modului de lucru intern. Microprocesorul controlează întregul PC şi foloseşte căi dedicate de control al informaţiilor denumite magistrale pentru a realiza trimiterea datelor către sub-sisteme, cum ar fi sistemul de memorie cache, memoria principală sau alte dispozitive ale calculatorului. Acestea reprezintă interfeţele externe ale microprocesorului şi pot avea structuri diferite chiar şi pentru unităţi de procesare altfel similare.

Există mai multe tipuri de magistrale ale unui calculator personal modern. Următoarea secţiune, referitoare la funcţiile magistralei de sistem ne oferă detalii specifice fiecărui tip de magistrală şi explicaţii generale legate de dimensiunile şi lăţimea diverselor tipuri magistrale. Putem găsi

aici informaţii interesante referitoare la magistrale sistem I/O de tipul PCI sau ISA. La nivelul microprocesorului, magistralele importante sunt magistrala procesorului şi cea de memorie, discutate mai în detaliu în secţiunea "Magistralele procesorului şi magistralele de memorie".

### *2.7.2 Magistrala sistem - funcţii şi caracteristici*

***Ierarhia de magistrală.*** Un calculator personal are o ierarhie de magistrale; orice calculator modern are cel puţin patru magistrale, considerate ierarhic, deoarece fiecare magistrală este, generalizând, parte a procesorului şi fiecare dintre acestea se conectează cu nivelul de deasupra, integrând unitar componentele PC-ului. De asemenea, magistralele respective nu lucrează la aceeaşi viteză, ci unele au viteze mai scăzute iar altele, în replică, mai mari - astfel, microprocesorul este cel mai rapid dispozitiv din interiorul unui calculator.

- *Magistrala procesorului*: reprezintă nivelul cel mai înalt de magistrală pe care cipul procesorului îl foloseşte pentru transferul informaţiilor de la şi înspre processor.

- *Magistrala cache:* arhitectură de nivel înalt, ca cele folosite în construcţia Pentium Pro şi Pentium II, ce implică o magistrală dedicată pentru accesarea memoriei cache. Aceasta este uneori denumită magistrala "din spate" (backside bus). Procesoarele convenţionale ce folosesc plăci de bază de generaţia a 5-a au memoria cache conectată la magistrala standard de memorie.

- *Magistrala de memorie:* este o magistrală de nivelul doi ce asigură conexiunea între sub-sistemul de memorie şi procesor; în unele sisteme magistralele procesor şi cea de memorie sunt practic acelaşi lucru.

- *Magistrala locală I/O:* este o magistrală de mare viteză de intrare/ieşire folosită pentru conectarea perifericelor la memorie şi procesor (spre exemplu: plăci video, dispozitive de stocare pe disc, interfeţe rapide de reţea, etc.). Cele mai uzuale astfel de magistrale sunt magistralele VESA Local Bus (VL-BUS) şi Peripheral Component Interconnect Bus (PCI).

- *Magistrala standard I/O:* conectată cu cele trei tipuri de magistrală descrise mai sus, folosindu-se vechiul standard de I/O, pentru periferice de viteză mai redusă, cum ar fi: mouse-uri, modemuri, plăci de sunet normale, plăci de reţea de viteză scăzută etc. Se foloseşte, de asemenea, pentru a se asigura compatibilitatea cu dispozitivele mai vechi. Această arhitectură este folosită la majoritatea PC-urilor actuale şi se numeşte arhitectură de

magistrală **ISA** (**I**ndustry **S**tandard **A**rchitecture) sau **EISA** (**E**nhanced **ISA** - ISA îmbunătățită).

Unele PC-uri mai noi folosesc un fel de magistrală adițională special proiectată pentru comunicații grafice; în fapt aceasta nu este chiar o magistrală, ci un port: portul de accelerare grafică (**AGP** - **A**ccelerated **G**raphics **P**ort). În general, diferența dintre un port şi o magistrală este aceea că magistrala este concepută pentru mai multe dispozitive ce îşi împart resursele mediului, în timp ce un port este utilizat doar pentru două dispozitive.

### *2.7.3 Magistralele procesorului şi magistralele de memorie*

Magistrala procesorului este dată de o mulțime de circuite menite să transporte informație de la şi înspre procesor, iar activitatea sa este controlată în mod normal de cipul sistemului. Magistrala de memorie asigură conexiunea dintre memoria principală şi memoria cache; în practică, aceste două magistrale pot fi considerate ca un tot unitar. De multe ori referirea se face la magistrala de memorie, care este, de fapt, principala caracteristică legată de date la nivelul de sistem al PC-ului. Toate transferurile de date spre (şi de la) procesor au loc folosindu-se această magistrală.

### *2.7.4 Magistrala de date dintre procesor şi memorie*

Fiecare magistrală este compusă din două părți distincte: magistrala de date şi magistrala de adrese. Magistrala de date reprezintă de fapt, ceea ce înțelegem de obicei când vorbim despre magistrale; aceste "canale" de transmisie ajută practic la transferul datelor în interiorul PC-ului. Cu cât magistrala de date este mai largă, cu atât mai multe date se pot transmite simultan, deci magistrale cu lărgimi de bandă mai mari implică în mod obişnuit performanțe superioare. Viteza magistralei este dată de viteza ceasului de sistem şi reprezintă o altă componentă deosebită în stabilirea performanțelor unui sistem de calcul.

Lărgimea de bandă a magistralei de date este dată deci de cantitatea de informații ce se scurge prin aceasta şi este în funcție de lățimea magistralei (în biți) şi de viteza ei (în MHz). Ne putem gândi la o magistrală ca la o autostradă: lățimea ei este dată de numărul de benzi, iar viteza este dată de cea a maşinilor care circulă pe aceasta. Lărgimea de bandă a

magistralei de memorie este extrem de importantă în PC-urile actuale, deoarece reprezintă de multe ori o strangulare a performanțelor calculatorului.

Deoarece procesoarele de astăzi rulează la viteze din ce în ce mai mari (de fapt mult mai mari decât alte părți ale sistemului), mărirea vitezei cu care se transferă datele către procesor pentru a se prelucra are o importanță deosebită asupra performanțelor generale ale calculatorului. De aceea, viteza microprocesorului nu mai afectează foarte mult performanțele globale ale calculatoarelor; spre exemplu, un Pentium 150 nu este cu mult mai rapid decât un Pentium 133 (P150 rulează pe o magistrală de 60 MHz de memorie iar P133 pe o magistrală de 66 MHz), astfel că o îmbunătățire cu 10% a vitezei magistralei are un efect mult mai mare decât o îmbunătățire cu 10% a vitezei microprocesorului.

### *2.7.5 Magistrala de adrese dintre procesor şi memorie*

Magistrala de adrese reprezintă canalele pe care se transportă informația despre locul în memorie unde datele sunt transferate, sau de unde se transferă. Această magistrală nu conține deci informație efectiv utilă, ci practic doar adrese de memorie ce determină locațiile unde datele sunt scrise sau de unde sunt citite.

Viteza magistralei de adrese este aceeaşi cu aceea a magistralei de date, iar lățimea acesteia determină "adresabilitatea" procesorului, adică cantitatea maximă de memorie ce poate fi accesată de către microprocesor. Nu întotdeauna lățimea magistralei de adrese coincide cu a celei de date şi nici nu se impune vreo restricție în acest sens; totuşi, calculatoarele actuale dispun de magistrale de date şi de adrese de lățimi suficient de mari pentru a asigura performanțe sporite de lucru. În general, mărimea magistralei de date nu reprezintă un concept foarte uzual atunci când vorbim despre caracteristicile unui calculator; poate datorită faptului că nu are un impact direct asupra performanțelor şi pentru că, de regulă, procesoarele pot adresa mult mai multă memorie (RAM) fizică decât folosesc marea majoritate a utilizatorilor de PC-uri. În plus față de aceasta, plăcile de bază ale calculatoarelor limitează dimensiunea maximă a memoriei RAM cu mult sub posibilitățile de adresare ale procesorului. De exemplu, un procesor Pentium normal poate adresa (teoretic) memorie până la limita de 4 GB, dar cele mai multe plăci de bază limitează această memorie la maximum un sfert din dimensiunea maximă (maxim 1 GB).

### *2.7.6 Mărimea magistralei de adrese pentru diverse procesoare*

La microprocesoarele din familia x86 lățimea de 32 de biți este cea mai uzuală (şi în mod implicit adresarea de maximum 4GB de memorie). Deoarece această limită de 4GB de memorie RAM este mai mult decât suficientă, după cum am exemplificat şi mai înainte, nu s-au făcut deci eforturi deosebite pentru a se mări. Doar la ultimile tipuri de microprocesoare, Pentium Pro, Pentium II, Pentium III şi Pentium IV lățimea magistralei de adrese a fost mărită la 36 de biți, de unde a rezultat deci o adresabilitate de 64 GB ($2^{36}$ octeți).

## 2.8 Ceasuri de sistem

Toate calculatoarele compatibile IBM-PC dispun de aşa numitele ceasuri de sistem sau, pe scurt, ceasuri. Un astfel de ceas este un dispozitiv ce emite impulsuri electrice la intervale regulate de timp, ca un metronom. Aceste impulsurile sunt emise cu o anumită frecvență, în mod normal măsurată în MHz (mega-hertzi, sau milioane de cicluri pe secundă).

Până la apariția procesorului 486, procesoarele calculatoarelor personale erau proiectate să folosească un ceas ce funcționa la dublul vitezei microprocesorului. Astfel, un procesor 386 la 25 Mhz folosea un ceas ce lucra la 50 MHz (această "regulă" era valabilă la toată familia de procesoare x86, de la Intel 8086 la 386). Începând cu 486, procesoarele au început să utilizeze aceeaşi frecvență cu cea a ceasului, fără s-o mai împartă la doi.

Un "tact" de ceas ("clock tick") reprezintă cea mai mică unitate de măsură în activitatea de procesare desfăşurată de microprocesor - acesta se mai numeşte şi "ciclu de ceas"; unele activități necesită un singur ciclu de ceas pentru a se termina, altele mai multe cicluri, în funcție de complexitatea acestora. Frecvența de tact a ceasului determină, deci, viteza cu care se desfăşoară activitățile în interiorul PC-ului. În acest sens, o viteză mai mare determină o performanță mai mare a calculatorului. Începând cu anul 1992, Intel a oferit soluții de creştere a vitezei de procesare a unui calculator fără a face modificări asupra plăcii de bază: procesoarele cu dublare (de exemplu, 486DX2/66) sau triplare (486DX4/100) a frecvenței ceasului.

Codul DX2 însemna dublarea frecvenței interne a ceasului, adică executarea instrucțiunilor din setul de bază al microprocesorului la o viteză

de două ori mai mare - în acest caz la 66 MHz față de 33 MHz, în timp ce DX4 înseamnă de fapt triplarea vitezei interne de execuție a instrucțiunilor (de la 33 MHZ la 100 MHz) şi nu cvadruplarea lor, cum am fi tentați să credem la prima vedere. De remarcat în ambele situații este faptul că funcționarea externă a procesorului - comunicarea cu magistralele, memoria, porturile, discurile etc. nu se modifică, ea este făcută la aceeaşi viteză: 33 MHz.

Vechile PC-uri dispuneau de un ceas de sistem unificat; un singur ceas ce rula la frecvențe foarte mici, de genul a 8 KHz, conduceau microprocesorul, memoria şi magistrala I/O.

Cu timpul, calculatoarele s-au modernizat şi unele dispozitive au câştigat în viteză față de altele; a apărut astfel necesitatea apariției de ceasuri diferite. Un calculator actual are de regulă patru sau cinci ceasuri ce rulează la viteze diferite (dar corelate). Atunci când se face referirea la "ceasul sistem", ne referim la viteza magistralei de memorie de pe placa de bază (şi nu la viteza internă a microprocesorului). Calculatoarele folosesc un singur circuit generator de tacturi (aflat pe placa de bază) pentru generarea tacturilor "ceasului sistem" şi apoi sunt folosite ceasuri multiplicatoare sau divizatoare pentru a crea alte semnale. Tabelul 2.8 de mai jos reflectă setarea ceasurilor într-un sistem PC Pentium II 266 MHz şi modul în care acestea sunt corelate.

Se observă din acest tabel că întregul sistem este bazat pe viteza ceasului sistem, de aceea creşterea vitezei ceasului sistem este mult mai importantă decât creşterea vitezei interne a microprocesorului; de fapt procesorul "pierde timp" în aşteptarea informațiilor venite de la dispozitive mult mai încete, cum ar fi magistralele de sistem.

Tabelul 2.8

**Setarea ceasului la un sistem Pentium II 266 MHz**

| Ceas Dispozitiv/Magistrală | Viteza (MHz) | Corelație |
|---|---|---|
| Microprocesor | 266 | Ceas sistem * 4 |
| Cache de nivel 2 | 133 | Ceas sistem * 2<br>(sau Procesor / 2) |
| Magistrala de sistem | 66 | - |
| Magistrale PCI | 33 | Ceas sistem / 2 |
| Magistrala ISA | 8.3 | Magistrală PCI / 4 |

## 2.9 Noțiuni de multiprocesare

Pe scurt, multiprocesarea înseamnă folosirea unui sistem de calcul ce conține două sau mai multe microprocesoare, în speranța că, folosind două (sau mai multe) procesoare în loc de unul, poți mări performanțele. Dar, cum este de aşteptat, nu totdeauna când se folosesc mai multe procesoare performanțele sunt mai bune, ci doar sub anumite condiții. Pentru a beneficia de avantajele multiprocesării un sistem de calcul trebuie să aibă:

- Placă de bază capabilă să suporte mai multe procesoare;
- Suport din partea procesoarelor - aceasta înseamnă că procesoarele trebuie să fie capabile să fie folosite într-un sistem multiprocesor. Nu toate procesoarele sunt capabile de acest lucru, unele variante de microprocesoare au suport pentru multiprocesare, altele nu;
- Suport din partea sistemului de operare - un sistem de operare ce suportă multiprocesarea, precum Windows NT/2000 sau UNIX.

În plus, multiprocesarea este efectiv eficientă atunci când aplicațiile software sunt proiectate în mod special pentru aceasta. Multiprocesarea este condusă de către sistemul de operare, ce alocă diferite sarcini *(task-uri)* pentru a fi soluționate de diferitele procesoare ale sistemului. Aplicațiile proiectate pentru multiprocesare sunt de regulă concepute astfel încât să poată fi divizate în mici subrutine care să poată fi rulate în mod independent. Aceasta permite sistemului de operare să asigure operarea acestor părți ale programului de către diferite procesoare în acelaşi timp, ceea ce reprezintă de fapt atuul multiprocesării. Dacă însă aplicația respectivă nu este concepută special pentru multiprocesare, nu putem beneficia de avantajele existenței a mai multor procesoare, deşi sistemul de operare poate face uz de procesoare adiționale atunci când rulăm mai multe aplicații în acelaşi timp (multitasking).

Multiprocesarea poate fi simetrică sau asimetrică, în funcție de modul în care sistemul de operare împarte sarcinile între procesoare. Multiprocesarea asimetrică este cazul în care unele procesoare execută doar sarcini de sistem iar altele rulează doar aplicații. Acest tip de multiprocesare este evident ineficientă în cazuri în care calculatorul trebuie să execute doar task-uri de sistem şi nici o aplicație utilizator, sau invers. Multiprocesarea simetrică *(SMP - Symmetric MultiProcessing)* permite execuția de task-uri de sistem sau utilizator de către orice procesor, ceea ce conduce la performanțe superioare. Acest tip de multiprocesare este folosită pe unele calculatoare personale.

Pentru ca un procesor să permită suport pentru multiprocesare, trebuie ca să accepte un protocol de multiprocesare ce va dirija modul în care procesoarele vor comunica unele cu altele în cadrul SMP. Procesoarele Intel Pentium Pro şi Pentium II folosesc un protocol SMP numit APIC, iar soclurile Intel ce suportă microprocesarea (precum 430HX, 440FX, 450GX/KX) sunt proiectate pentru a lucra cu aceste cipuri. APIC este un standard patentat de Intel, pentru a preveni firmele concurente Cyrix şi AMD pentru a implementa APIC, ceea ce înseamnă că, deşi AMD şi Cyrix produc procesoare compatibile Intel, nu pot să le facă să lucreze în configurații SMP pe plăci de bază cu socluri standard Intel; deci din punctul de vedere al multiprocesării simetrice, Intel şi-a rezervat întreaga piață de calculatoare.

Firmele AMD şi Cyrix au implementat propriul standard de multiprocesare simetrică, numit OpenPIC. AMD oferă suport pentru standardul OpenPIC cu soclul AMD-640 şi permite multiprocesare simetrică începând cu procesorul K6. Procesorul Intel Pentium Pro a reprezentat cea mai bună soluție la un moment dat pentru multiprocesare simetrică deoarece fiecare cip are memorie cache de nivel 2 integrată. Într-un sistem cu mai mult de un procesor şi memorie cache de nivel 2 pe placa de bază, procesoarele trebuie să împartă această memorie. Fiecare nou procesor introdus va determina o scădere a memoriei cache pentru celelalte procesoare, ceea ce va scădea performanțele. Deoarece fiecare Pentium Pro are memorie cache de nivelul 2 proprie, problema partajării memoriei cache între procesoare dispare şi performanțele sunt mult îmbunătățite.

## 2.10 Funcționarea şi operarea întreruperilor

### *2.10.1 Generalități referitoare la întreruperi*

Microprocesorul este un dispozitiv bine reglat pentru a îndeplini (în mod normal) un anumit lucru la un moment dat. Totuşi, atunci când folosim un computer, este necesar ca microprocesorul să facă mai multe lucruri în acelaşi timp (sau cel puțin să lase să pară acest lucru). De exemplu, pentru orice utilizator de Windows 9x, NT, 2000, XP (sistem de operare multitasking) este normal ca în timp ce editează un document să asculte muzică de la un CD-ROM sau să încarce un fişier prin modem de pe Internet.

Procesorul este capabil să execute toate aceste sarcini aparent simultan prin mecanismul de "time-sharing", adică împărțindu-se între diversele dispozitive ce îi solicită activitatea. Numai viteza extraordinară a microprocesorului de comutare între diferitele sarcini pe care le are de îndeplinit ne dă de fapt iluzia că acestea sunt executate în acelaşi timp.

Majoritatea componentelor unui calculator trebuie să schimbe informații cu microprocesorul şi este de aşteptat să capteze atenția acestuia atunci când este nevoie. Procesorul are astfel rolul de a balansa transferul de informații de la componentele calculatorului şi de a le organiza într-o manieră eficientă. Există două căi de bază în îndeplinirea acestor sarcini, şi anume:

- *prin sondare (polling)*. Procesorul poate trimite mesaje dispozitivelor calculatorului pentru a le "întreba" dacă au nevoie de ceva. Această metodă nu este însă folosită într-un PC, pe baza unor motive întemeiate. Primul motiv ar fi acela că se face o risipă de timp pentru a verifica fiecare dispozitiv în parte dacă are nevoie de acțiunea procesorului; altă motivație ar fi aceea că diversele componente ale calculatorului au nevoie de colaborarea procesorului în procente diferite, de exemplu, mouse-ul are nevoie de mult mai puțină atenție decât, să spunem, un hard-disk ce este în activitate.

- *prin întrerupere (interrupting)*. Cealaltă modalitate de balansare a transferului de informații de la componente este aceea de a le lăsa pe acestea sâ ceară "atenția" atunci când au nevoie. Astfel au apărut întreruperile. Atunci când un dispozitiv trebuie să transfere date către microprocesor, generează o întrerupere către acesta, ceea ce este, de fapt, un mesaj de genul: *"Am nevoie de atenția dvs., vă rog"*. În acest moment microprocesorul se întrerupe din activitate şi îndeplineşte cererea dispozitivului care i-a cerut atenția. Aceste cereri sunt rezolvate pe baza unei liste de priorități, pentru a se decide care este mai importantă şi deci, care va fi executată prima.

Poate părea la prima vedere puțin ineficient ca procesorul să fie întrerupt tot timpul funcționării calculatorului. Este destul de neplăcut să stai de pildă la servici şi să fii sunat din cinci în cinci minute de către cineva şi să nu apuci să termini ce ai de lucru. Să ne gândim totuşi, la varianta în care la fiecare 30 de secunde am fi nevoiți să sunăm pe cineva pe care să-l întrebăm dacă are nevoie de noi... Din fericire, microprocesorul nu este o ființă umană şi poate fi "deranjat" în acest mod, prin întreruperi.

În afară de întreruperile hardware despre care discutăm aici, există şi întreruperi software, care sunt folosite de numeroase produse program ca

răspuns la diverse evenimente ce apar în timpul rulării sistemului de operare sau al aplicațiilor. În esență, acestea generează întreruperea microprocesorului de către el însuşi! Acest lucru face parte din modul în care un procesor este în stare să facă mai multe lucruri o dată. Alt lucru pe care îl fac întreruperile software este acela că permit unui program să acceseze alt program fără să ştie unde se află în memorie.

După cum am mai spus, procesorul dispune de capacitatea de a fi întrerupt, dar şi de posibilitatea de a conserva activitatea întreruptă cât timp se prelucrează cerearea solicitată. În acest sens este folosită stiva microprocesorului, care joacă rorlul de depozit pentru toate informațiile curente despre activitățile calculatorului. Atunci când se execută o subrutină sau se întrerupe temporar o activitate pentru a se focaliza asupra alteia, informațiile despre locul unde se afla şi ce făcea calculatorul sunt depozitate în stivă astfel încât să se poată întoarce la activitatea întreruptă. Când se trece la o nouă activitate, informațiile despre aceasta vor fi plasate în vârful stivei, îndicând activitatea curentă. Când activitatea calculatorului se termină, informațiile despre activitatea respectivă sunt eliminate din stivă, iar sarcina anterioară îşi reia locul în vârful stivei, devenind astfel noua sarcină curentă, iar calculatorul revenind în acest mod la operația anterioară.

Viteza procesorului este de asemenea, fascinantă, mai ales la calculatoarele ultimelor generații. Există astfel termenul de *MIPS (Millions of Instructions Per Second - Milioane de Instrucțiuni pe Secundă),* care inițial era folosit doar pentru calculatoarele de tip mainframe (de performanțe net superioare calculatoarelor personale), dar, datorită creşterii remarcabile a performanțelor PC-urilor, se utilizează în prezent şi pentru acestea. Dacă primele cipuri Intel atingeau o treime de *MIPS*, procesoarele Pentium actuale depăşesc *150 MIPS*. Altă unitate uzuală pentru măsurarea vitezei procesorului este MHz, despre care am mai discutat în cursul acestui capitol. Reamintim că primul 8086 rula la o fecvență de 4,77 MHz, în timp ce procesoarele actuale Pentium rulează până la o frecvență ce depăşeşte 3000 MHz. Viteza de procesare a cipului este într-adevăr fascinantă: spre exemplu, un procesor la numai 200 MHz execută 20.000.000 de instrucțiuni între fiecare apăsare a tastelor pe care o facem în decursul unei redactări (rapide) de text, iar în cazul transferului de date cu un modem de 28.800 KB/sec procesorul execută în jur de 60.000 de instrucțiuni între doi octeți care se transferă.

### *2.10.2 Controllere de întreruperi*

Întreruperile generate de dispozitivele din interiorul calculatorului sunt aduse la microprocesor folosind o componentă aparte hardware denumită controller de întreruperi. Standardul în acest domeniu este controller-ul de întreruperi Intel 8259 şi datează de la primele PC-uri. Ca în majoritatea cazurilor legate de controllere dedicate, plăcile de bază moderne controllerul 8259 este încorporat într-un cip mai mare ca parte a unui circuit. Controlerul de întreruperi are opt linii de intrare ce primesc cererile de la unul din cele opt dispozitive diferite, apoi trimite cererea microprocesorului, spunându-i ce dispozitiv a lansat acea cerere (ce număr de întrerupere are cererea, de la 0 la 7). Primul PC avea un singur astfel de controller şi deci suporta doar întreruperi de la 0 la 7.

Începând cu calculatorul IBM-AT a fost adăugat al doilea controller de întreruperi pentru extinderea sistemului, reprezentând extinderea magistralei sistem *ISA (Industry Standard Architecture)* de la 8 la 16 biţi. Pentru a se păstra însă compatibilitatea cu sistemele mai vechi, proiectanţii sistemelor AT nu au înlăurat vechea linie de întreruperi către procesor, ci au folosit (în cascadă) ambele controllere împreună. Primul controller de întreruperi are încă opt intrări şi o singură ieşire către microprocesor, în timp ce al doilea are aceeaşi schemă, dar i se adaugă opt noi intrări (dublând în acelaşi timp numărul de întreruperi) iar ieşirea sa alimentează intrarea numărul 2 a primului controler. Astfel, dacă o intrare a celui de-al doilea controller devine activă, ieşirea sa declanşează întreruperea 2 a primului controller, care trimite un semnal către microprocesor. De aceea, această modalitatea de funcţionare a celor două controllere se numeşte funcţionare în cascadă.

Ce se întâmplă atunci cu întreruperea 2? Această linie de întrerupere *(IRQ - Interrupt ReQuest Line)* este folosită acum pentru a asigura funcţionarea în cascadă a celui de-al doilea controller, astfel încât proiectanţii sistemului AT au redirecţionat circuitul plăcii de bază al IRQ2 către IRQ9, deci dacă vreun dispozitiv al sistemului încearcă să folosească IRQ2, el foloseşte, în realitate, întreruperea 9 (IRQ9).

Dispozitivele proiectate să folosească IRQ2 ca setare iniţială sunt relativ rare în ziua de astăzi, din moment ce întreruperea 2 a fost scoasă din uz de peste zece ani. În cele mai multe cazuri IRQ2 se socoteşte inutilizabilă, în moment ce IRQ9 este o linie de întreruperi perfect valabilă. Există totuşi unele modemuri care utilizează întreruperea 2 pentru a ocoli faptul că porturile COM3 şi COM4 partajează întreruperi cu COM1 şi

COM2 în absența altor setări. În cazul existenței într-un PC a mai multor dispozitive ce folosesc întreruperi IRQ1-IRQ8, este necesar să folosim uneori un astfel de modem setat pentru IRQ2, care foloseşte de fapt IRQ9; în acest caz trebuie să fim atenți pentru a nu a avea alt dispozitiv setat pentru a folosi IRQ9 (ar intra astfel în conflict cu modemul).

### *2.10.3 Linii IRQ şi magistrala sistem*

Dispozitivele ce folosesc întreruperi le semnalează prin liniile de întrerupere ale magistralei ISA. Majoritatea întreruperilor cu care este înzestrată magistrala de sistem este pentru uzul diverselor dispozitive; totuşi unele dintre acestea sunt folosite intern de către sistem. Din această cauză nu există circuite fizice pe magistrală - acestea sunt întreruperile 0, 1, 2, 3, 8 şi 13 şi nu sunt disponibile pentru alte dispozitive externe (plăci de extensie, modemuri etc.). Toate aceste considerații sunt valabile pentru plăcile ISA şi VESA, plăcile PCI tratând întreruperile în mod diferit, prin propriul sistem de întreruperi interne.

***Prioritatea întreruperilor.*** Tratarea întreruperilor în interiorul PC-ului se face corespunzător nivelelor de priorități asociate acestora. Aceste priorități sunt în funcție de linia de întreruperi pe care o folosesc pentru a intra în controller-ul de întreruperi. Din acest motiv, nivelele de priorități sunt direct legate de numărul întreruperii:

- În cazul vechilor PC-XT, prioritățile întreruperilor sunt 0, 1, 2, 3, 4, 5, 6, 7.
- În cazul unui PC modern, este un pic mai complicat, având în vedere că cel de-al doilea set de întreruperi este direcționat prin canalul IRQ2 al primului controller. Asta înseamnă că primul controller "vede" aceste întreruperi cu prioritatea întreruperii doi, ceea ce ne conduce la următorul şir de priorități: 0, 1, (8, 9, 10, 11, 12, 13, 14, 15), 3, 4, 5, 6, 7. În decursul oricărui eveniment, nivelul de prioritate al întreruperilor nu influențează în mare măsură performanțele (viteza) calculatorului, ele practic există pentru ca procesorul să ştie în ce ordine să le examineze, pentru a nu se genera conflicte.

### *2.10.4 Întreruperi nemascabile*

Întreruperile uzuale pe care le folosim în mod normal şi le referim printr-un număr se numesc întreruperi mascabile. Microprocesorul este capabil să "mascheze" - aceasta înseamnă, de fapt, să ignore temporar - o întrerupere de acest gen, pentru a termina alt proces început anterior. Pe lângă aceste întreruperi mascabile, există şi întreruperile nemascabile *(NMI - Non Maskable Interrupts)* care nu pot fi "mascate" (ignorate) şi se folosesc atunci când condiţii excepţionale necesită atenţia imediată a procesorului. Întreruperile nemascabile nu pot fi ignorate de către procesor decât dacă acest lucru este specificat în mod direct.

Atunci când este recepţionat un semnal NMI, procesorul se opreşte din activitate indiferent care ar fi aceasta şi acordă atenţia acestui semnal. Deoarece această modalitate de lucru este destul de "dură", în sensul că procesorul trebuie să renunţe necondiţionat la activitatea curentă şi să se ocupe de întreruperea NMI, aceste întreruperi nemascabile sunt folosite doar pentru probleme absolut critice, cum ar fi erorile hardware. Cea mai des uzitată întrerupere NMI este aceea pentru controlul parităţii memoriei; dacă se semnalează o astfel de eroare trebuie să fie imediat remediată pentru a se preveni eventualele pierderi de date - în acest caz se sistează funcţionarea calculatorului (halt), pentru a se corecta defecţiunea hard apărută.

În tabelul 2.9 sunt prezentate o serie dintre cele mai uzuale întreruperi şi rolul acestora.

Tabelul 2.9

**Un sumar al întreruperilor şi utilizarea lor obişnuită**

| IRQ | Linie BUS | Prioritate | Utilizare obişnuită | Alte utilizări |
|---|---|---|---|---|
| 0 | - | 1 | Timerul sistem | - |
| 1 | - | 2 | Controller-ul de tastatură | - |
| 2 | - | - | Nici una; înlocuită de IRQ 9 | Modemuri, plăci video EGA, COM3, COM4 |
| 3 | 8/16 biţi | 11 | COM2 | COM4, modemuri, plăci de sunet, plăci de reţea, plăci acceleratoare de bandă |
| 4 | 8/16 biţi | 12 | COM1 | COM3, modemuri, plăci de sunet, plăci de reţea, plăci acceleratoare de bandă |
| 5 | 8/16 biţi | 13 | Placa de sunet | LPT2, LPT3, COM3, COM4, modemuri, plăci de reţea, plăci acceleratoare de bandă, controllere de HD la vechile PC-XT |
| 6 | 8/16 biţi | 14 | Controller floppy disk | Plăci acceleratoare de bandă |

| IRQ | Linie BUS | Prioritate | Utilizare obişnuită | Alte utilizări |
|---|---|---|---|---|
| 7 | 8/16 biţi | 15 | LPT1 | LPT2, COM3, COM4, modemuri, plăci de sunet, plăci de reţea, plăci acceleratoare de bandă |
| 8 | - | 3 | Ceasul în timp real | - |
| 9 | 16 biţi | 4 | - | Plăci de reţea, plăci de sunet, adaptoare gazdă SCSI, dispozitive PCI, dispozitive IRQ2 redirectate |
| 10 | 16 biţi | 5 | - | Plăci de reţea, plăci de sunet, adaptoare gazdă SCSI, canalul IDE secundar, canalul IDE cuaternar, dispozitive PCI |
| 11 | 16 biţi | 6 | - | Plăci de reţea, plăci de sunet, adaptoare gazdă SCSI, plăci video VGA, dispozitive PCI, canalul IDE terţiar şi cuaternar |
| 12 | 16 biţi | 7 | PS/2 mouse | Plăci de sunet, de reţea, adaptoare gazdă SCSI, plăci video VGA, canalul IDE terţiar, dispozitive PCI |
| 13 | - | 8 | FPU/NPU/Coprocesor matematic | - |
| 14 | 16 biţi | 9 | Canalul IDE primar | Adaptoare gazdă SCSI |
| 15 | 16 biţi | 10 | Canalul IDE secundar | Plăci de reţea, adaptoare gazdă SCSI |

## 2.11 Procesoarele RISC

### *2.11.1 Introducere*

Termenul de procesor *RISC (Reduced Instruction Set Computer - Calculator cu Set Redus de Instrucţiuni)* a mai fost folosit pe parcursul acestui capitol, în comparaţie cu procesoarele *CISC (Complex Instruction Set Computer - Calculator cu Set Complex de Instrucţiuni);* a sosit momentul să dăm mai multe detalii referitoare la acest subiect şi să explicăm diferenţele dintre procesoarele RISC şi CISC.

Procesoarele RISC, apărute pentru prima oară în anii 80, păreau la început predestinate să domine industria computerelor în anii 90 şi să facă uitate vechile arhitecturi de calculatoare. Practic toţi producătorii importanţi din industria calculatoarelor oferă acum sisteme gen RISC; giganţii IBM şi Hewlett Packard şi-au dezvoltat propriile procesoare RISC, în timp ce alţi producători, ca DEC sau Siemens, au preferat să cumpere licenţe ale unor arhitecturi deja existente pentru a ţine pasul cu concurenţa acerbă din domeniu.

Conceptul de arhitectură "RISC" este deseori greşit utilizat sau definit, de aceea, pentru a fi definit şi înțeles mai bine, trebuie să facem o întoarcere în timp şi să vedem, de asemenea, diferențele esențiale între cele două noțiuni: CISC şi RISC. Era recunoscut de prin anii 50 faptul că se putea sacrifica din eficiența memoriei la codificarea unui set de instrucțiuni pentru a câştiga în performanță. Instrucțiunile simple, de lungime fixă, erau uşor de implementat şi se executau mai rapid. Această tehnică era folosită pentru implementarea setului de instrucțiuni al calculatorului IBM 360 de tip *mainframe* din anii 50-60. Acest set de instrucțiuni se baza pe o arhitectură clasică CISC, dar mecanismul de microcod ce execută de fapt instrucțiunile era un procesor RISC mai simplu.

Microcodul este de fapt, software-ul de nivel jos care conduce execuția setului de instrucțiuni, iar procesoarele RISC se numeau controllere orizontale de microcod. Cu toate că erau cunoscute avantajele arhitecturilor RISC, costurile ridicate ale memoriei determinau folosirea în continuare a arhitecturilor CISC, mai eficiente din punct de vedere al capacității de stocare şi care păreau că reprezintă cea mai bună soluție în acel moment (se utilizau instrucțiuni capabile să facă mai multe lucruri).

Primele inovații față de vechile arhitecturi de microprocesor au apărut în cadrul firmei IBM, în cadrul unui proiect început în 1975 şi care acum se consideră a fi pionieratul în domeniul arhitecturii RISC. John Cocke, un inginer de la IBM, a observat că doar o mică parte (aproximativ 10%) a mulțimii de instrucțiuni a calculatorului IBM 360 era folosită în majoritatea timpului, iar această submulțime concentra cel mai mare procent din timpul de execuție (90%). Membrii echipei IBM şi-au propus astfel să simplifice mulțimea de instrucțiuni pentru a obține o medie de o execuție pe ciclu de ceas. Acest obiectiv era realizabil doar dacă mulțimea de instrucțiuni era structurată în conductă, salvându-se în acest mod timpul pierdut pentru aducerea şi decodarea instrucțiunilor din memorie.

Două noi proiecte ce au pornit câțiva ani mai târziu au adus conceptele RISC în centrul atenției arhitecturilor de computere. Primul dintre acestea a fost condus de David Patterson de la Universitatea din Berkeley şi a culminat cu definiția procesoarelor RISC I şi RISC II la începutul anilor 80. Patterson a conturat, de asemenea, conceptul RISC. Potrivit acestuia, procesoarele RISC au inaugurat o nouă mulțime de principii arhitecturale. Din această cauză, noțiunea de RISC a fost considerată mai degrabă o filozofie decât o rețetă arhitecturală diferită. Punctele relevante ale filozofiei proiect menționate de Patterson sunt:

- mulțimea de instrucțiuni trebuie să fie simplă;

- instrucțiunile trebuie să ruleze la cea mai mare rată posibilă;
- noțiunea de "pipelining" este mai importantă decât mărimea programului;
- tehnologia compilatorului este un element critic într-un proiect RISC: optimizarea compilatoarelor trebuie să translateze cât mai mult posibil din complexitatea hardware-ului către faza de compilare.

Rezultatele acestor cercetări au dat naştere unei arhitecturi mai simple, caracterizată de instrucțiuni mai puține, multe registre, acces simplificat pentru încărcarea şi depozitarea datelor în memoria principală şi posibilitatea execuției instrucțiunilor într-o singură perioadă de ceas. Procesorul respectiv era mai mic, cu performanțe mai mari, dar se folosea cu 20-25% mai multă memorie şi erau necesare memorii cache scumpe pentru a ține "ocupat" microprocesorul RISC. Din această cauză, costurile ridicate ale arhitecturilor RISC au împiedicat răspândirea acestora pe piața consumatorilor medii şi a aplicațiilor comerciale. Procesoarele RISC erau însă folosite uzual în stațiile de lucru foarte puternice pentru aplicații ştiințifice, tehnice şi militare, unde se justificau prețurile mari pentru performanțe înalte.

O dată cu evoluția microprocesoarelor RISC, s-a descoperit că avantajul acestora nu costă în micşorarea setului de instrucțiuni, ci în simplitatea acestora. În zilele de astăzi majoritatea microprocesoarelor RISC au cam acelaşi număr de instrucțiuni ca şi cele CISC. Datorită modurilor mai simple de adresare ale instrucțiunilor RISC, având nevoie de un singur acces la memoria principală şi putând fi executate într-un singur ciclu de ceas, execuția lor a putut fi foarte uşor implementată în structuri de tip pipeline şi structuri superscalare ce permit execuția simultană a mai multor instrucțiuni.

Tot evidențiind avantajele tehnologiei RISC, se pune în mod evident întrebarea: De ce să mai folosim procesoare CISC, când cele RISC sunt în mod clar mai performante?

Pentru a răspunde la această întrebare, să evidențiem câteva aspecte ale problemei. În primul rând, diferențele dintre microprocesoarele CISC şi cele RISC nu mai sunt aşa de mari odată cu implementările RISC făcute în ultimii ani de către microprocesoarele compatibile Intel. Avantajul major care apare prin folosirea procesoarelor CISC este acela al compatibilității soft; astfel, cantitatea de software ce rulează în prezent pe microprocesoare CISC este imensă şi deocamdată nu se poate renunța la ea. Sucesul deosebit al sistemului de operare Windows 95 (ce rulează pe procesoare CISC) şi

faptul că noua sa versiune Windows 98, va rula tot pe platforme cu procesoare CISC, face să se mențină încă utilizarea îndeosebi a acestor tipuri de microprocesoare. Dacă cineva doreşte să achiziționeze un calculator bazat pe un microprocesor RISC, nu va putea beneficia de programele scrise pentru calculatoarele PC, deoarece majoritatea acestui soft este special proiectat pentru calculatoare PC.

Dacă softul pentru PC va dori să ruleze pe un procesor RISC, vor apărea numeroase probleme, printre care:

- aplicațiile au fost compilate pentru a lucra doar cu setul de instrucțiuni x86;
- softul se aşteaptă să sesizeze un mediu de operare Microsoft sub care să lucreze;
- multe aplicații şi jocuri DOS mai vechi trebuie să găsească mediul hard al calculatorului PC, lucrând de multe ori direct cu resursele hard ale calculatorului.

Pentru prima problemă ar putea exista soluția recompilării aplicației astfel încât să poată opera cu setul de instrucțiuni al microprocesorului RISC. La ora actuală, multe aplicații sunt disponibile în mai multe versiuni, fiind compilate pentru platforme RISC, dar numărul acestora este totuşi destul de redus şi există rețineri în privința firmelor de a elabora mai multe versiuni (din acest punct de vedere) ale aceleiaşi aplicații. Reținerile acestor firme sunt întemeiate, deoarece piața este prea mică pentru ca ele să-şi permite o asemenea dezvoltare şi, în al doilea rând, este greu de menținut un nivel apropiat pentru două sau mai multe versiuni de program.

În concluzie, apare o problemă cu dublu sens: nu există multe calculatoare RISC pentru că nu există soft pentru ele şi nu există soft pentru că nu există calculatoare RISC!

Pentru a doua problemă, s-ar părea că firma Microsoft a rezolvat problema odată cu apariția sistemului de operare NT care oferă portabilitate pentru procesoarele MIPS şi ALPHA de tip RISC. Inexistența unui sistem de operare Microsoft este o piedică esențială în calea pătrunderii procesoarelor RISC pe piața calculatoarelor personale.

Dacă pentru primele două probleme prezentate mai sus există soluții de rezolvare, pentru cea de-a treia nu există o soluție tehnică generală, din moment ce aplicațiile scrise pentru un mediu hardware specific unui PC nu vor putea rula pe procesoare RISC. În acest sens, singura soluție viabilă este practic rescrierea aplicației pentru noua platformă, soluție care nu se impune

din aceleaşi considerente prezentate mai înainte: numărul de staţii de lucru ce folosesc microprocesoare RISC este încă redus.

### *2.11.2 Tipuri de procesoare RISC*

Cele mai importante arhitecturi ce conţin procesoare RISC sunt:

- MIPS, folosite în staţii de lucru Silicon Graphics;
- SPARC, folosite în staţii de lucru Sun;
- PA-RISC, folosite în staţii de lucru Hewlett-Packard;
- PowerPC, folosite în calculatoare IBM PC şi Apple Macintosh;
- Alpha, folosite în staţii de lucru DEC (Digital Equipment Corporation).

Competiţia pe piaţa microprocesoarelor RISC este foarte mare, tot timpul apărând noi versiuni de procesoare. Toate aceste arhitecturi evoluează însă în aceeaşi direcţie:

- implementare pe 64 de biţi;
- unităţi performante de execuţie;
- noi instrucţiuni pentru aplicaţii multimedia şi DSP;
- frecvenţe de ceas intern foarte mari, superioare procesoarelor CISC;
- implementări superscalare, putând să execute mai multe instrucţiuni simultan;
- unităţi de operare în virgulă mobilă foarte puternice;
- memoria cache integrată de dimensiuni mari.

#### *2.11.2.1 Procesoarele din seria MIPS (Silicon Graphics)*

La originea acestor microprocesoare se află nişte proiecte experimentale iniţiate la Universitatea din Stanford la începutul anilor 80. Traducerea termenului "MIPS" ne oferă o imagine relevantă asupra filozofiei proiectului respectiv: *MIPS* provine de la *Microprocessor without Interlocking Pipeline Stages (Microprocesor fără stadii în conductă blocate)*. Obiectivul proiectanţilor MIPS a fost acela de a produce un procesor RISC cu funcţionare în conductă şi interblocare pipeline controlate software. Dacă o instrucţiune necesită două cicluri de ceas pentru a fi executată, este de datoria compilatorului să programeze o instrucţiune de

tipul NOP (No OPeration) următoare. În acest mod singura modalitate prin care se întrerupe funcționarea normală în timpul execuției sunt aceste instrucțiuni NOP controlate software (de compilator), în timp ce partea hardware nu va bloca de fiecare dată execuția pipeline. Această caracteristică reduce cantitatea de componente hardware necesare pentru manufacturarea procesorului.

Un produs MIPS din anul 1995 a fost MIPS T5 (redenumit apoi R1000), cu o arhitectură superscalară pe 64 de biți nouă, compatibilă cu cipurile mai vechi Rxxx. Arhitectura scalară dispunea de cinci canale, 64 de registre interne şi o memorie cache internă de 32 KB, utilizându-se o tehnologie de fabricație de 0,35 de microni.

Unele concepte deosebit de interesante cu privire la acest aspect au fost studiate la Universitatea Stanford cu MIPS-X, un produs derivat al arhitecturii MIPS ce avea o serie de caracteristici în plus. Multe dintre acestea au fost mai târziu introduse în procesorul comercial MIPS. Microprocesorul MIPS R2000 este un procesor pe 32 de biți cu o memorie cache de nivel 2, diferențiată pentru instrucțiuni şi date. O memorie tampon de scriere ajută la manipularea tututor datelor stocate în memorie. Produsul R2000 foloseşte o magistrală comună pentru memoria cache externă – o arhitectură non Harvard (reamintim că arhitectura Harvard presupune utilizarea de magistrale diferite pentru instrucțiuni şi pentru date). Construcția acestui procesor înglobează o arhitectură radicală de coprocesor. Unitatea de control a întregilor din UCP este separată de aşa numitul „Coprocesor de control al sistemului" (System Control Coprocessor), care este, de fapt, un controlor de memorie cache integrat direct pe cip. UCP şi unitatea de calcul în virgulă mobilă comunică prin intermediul memoriei. Microprocesorul înglobează 32 de regiştri generali şi 16 regiştri (pe 64 de biți) separați pentru calcule în virgulă mobilă. Coprocesorul pentru calculul în virgulă mobilă coține o unitate pentru adunare, una pentru împărțire şi una pentru înmulțire. Nu există biți de testare a condițiilor (indicatori de stare, sau *flags*, cum sunt denumiți la Intel). Programarea regiştrilor este controlată software.

### *2.11.2.2 Procesoarele din seria SPARC (Sun Microsystems)*

Procesorul SPARC *(Scalable Processor ARChitecture)* se poate "lăuda" ca fiind descendentul unei familii ilustre de microprocesoare, aceea a procesoarelor RISC-I şi RISC-II dezvoltate la Universitatea din Berkeley

în anii 80. Această arhitectură a fost definită de firma Sun Microsystems şi actualizată în permanență. Firma Texas Instruments a fost unul dintre principalii furnizori de cipuri ca urmare a unui contract cu firma Sun, unul dintre produsele anului 1995 fiind UltraSPARC, cu o arhitectură pe 64 de biți şi o implementare superscalară cu patru canale. O caracteristică importantă a arhitecturii este adăugarea de noi instrucțiuni pentru accelerarea graficii şi a prelucrărilor video; astfel pot fi prelucrați până la opt pixeli într-o singură instrucțiune sau ciclu de ceas. Dacă, în general, arhitectura acestui procesor este o arhitectură de tip RISC, există două "curiozități" ale acesteia, care îl disting în familia procesoarelor RISC.

În primul rând, SPARC utilizează conceptul de "ferestre de registre" (*register windows*) în scopul eliminării operațiilor de încărcare şi stocare în stivă ce apar la apelurile de proceduri. Acest lucru putea fi însă obținut şi prin programarea regiştrilor în momentul compilării. Echipa de la Berkeley a utilizat însă aceste ferestre de registre deoarece nu avea la momentul respectiv expertiza (pentru creearea compilatorului) necesară pentru a implementa alocarea interprocedurală a regiştrilor cu ajutorul software-ului (compilatorului).

În al doilea rând, o altă curiozitate a arhitecturii SPARC o reprezintă existența instrucțiunilor "etichetate" (*tagged instructions*). Se ştie că limbaje de programare declarative de genul Lisp sau Prolog folosesc tipuri de date etichetate. Arhitectura SPARC utilizează instrucțiuni ce pot manipula cu uşurință o etichetă (în engleză *tag*) pe 2 biți în fiecare cuvânt de memorie. Această caracteristică putea mări viteza de execuție a unui program Lisp cu câteva procente.

Procesoarele UltraSPARC IV sunt procesoare ce suportă două fire de execuție (*chip multithreading*) pe procesor, bazate pe două stadii pipeline UltraSPARC III. Alte caracteristici:

- 66 milioane de tranzistori pe cip;
- pipeline cu 14 stadii;
- frecvența de ceas între 1.05-1.2 GHz;
- L1 cache de 64KB pentru date şi 32 KB pentru instrucțiuni, 2KB Write, 2KB Pre-fetch;
- L2 cache de 16 MB;
- scalabilitate multiprocesor cu suport arhitectural până la 1000 de procesoare pe un singur sistem;
- controller-ul de memorie este capabil să adreseze până la 16 GB de memorie principală la o viteză de 2,4 GB/s.

Sun Microsystems Inc. este cel mai titrat producător de procesoare ce utilizează mulţimea de instrucţiuni SPARC, dar nu este singurul producător. Alt producător important este Fujitsu, ale cărui noi procesoare Sparc64 VI cu nume de cod Olympus vor veni pe piaţă în 2005, la viteze de peste 2,4 GHz, manufacturaţi în tehnologie de 90 de nm (nanometri). Performanţa estimată a acestui procesor este de patru ori mai mare decât a generaţiei actuale de la Fujitsu, SPARC64 V, ce rulează la 1,35 GHz.

### *2.11.2.3 Procesorul PA-RISC (Hewlett Packard)*

Arhitectura *PA-RISC (Precision Architecture)* a firmei Hewlett-Packard este destinată staţiilor de lucru performante, adoptând o linie nouă şi modernă. Performanţele de operare în virgulă mobilă ale acesteia sunt excelente faţă de majoritatea competitorilor. S-au inclus noi instrucţiuni pentru funcţii de accelerare a graficii şi a procesărilor video, similare celor de la SPARC. Numărul de formate de instrucţiuni este mai mare decât la orice alt procesor RISC: sunt prezente nu mai puţin de 12 combinaţii diferite de *opcode* (coduri de operaţie) şi regiştri sau câmpuri pentru constante într-un singur cuvânt (spre comparaţie, procesoarele SPARC şi MIPS pot utiliza doar patru combinaţii diferite).

Există în mod normal două moduri diferite de adresare, precum şi încă două moduri adiţionale ce oferă suport pentru operaţiile ce au loc înainte sau după modificarea unui registru index. Acest lucru oferă posibilitatea utilizării în total a patru modalităţi de adresare. Arhitectura PA (Precision Architecture) posedă coduri de operaţii (*opcode*) pe 6 biţi. Acest lucru reduce numărul de instrucţiuni posibile la mai puţin de 64 ($2^6$), deşi anumite instrucţiuni au mai multe variante, folosind biţi speciali în cadrul formatului instrucţiunii. Numărul de regiştri generali este de 32, completaţi cu încă 32 de regiştri cu caracter special, utilizaţi pentru administrarea întreruperilor, a nivelurilor de protecţie etc. Caracteristica atipică a acestui procesor este aceea că implementarea execuţiei în *pipeline* se face pe doar trei nivele, iar funcţionarea optimă a conductei necesită programare software.

### *2.11.2.4 Procesoarele PowerPC (IBM şi Motorola)*

Susţinut de firme puternice, ca IBM, Apple şi Motorola, PowerPC este concurentul principal al microprocesoarelor bazate pe arhitectura x86.

Principalul avantaj constă în posibilitatea rulării software-ului Apple, PC şi Unix. Folosind tehnicile de recompilare binară, integrând şi un emulator rapid pentru x86, procesorul PowerPC este capabil să utilizeze majoritatea sistemelor de operare şi a software-ului într-un singur sistem.

### *2.11.2.5 Procesoarele Alpha – DEC (Digital)*

Aceste procesoare se deosebesc de celelalte procesoare RISC prin fecvenţele foarte mari ale ceasului intern, arhitectura modernă pe 64 de biţi a acestora fiind una dintre cele mai performante de pe piaţă.

### *2.11.3 Concluzii şi viitorul procesoarelor RISC*

Fără îndoială, de la apariţie şi până în prezent, microprocesoarele RISC au avut performanţe superioare celor bazate pe arhitectura CISC. În ultimii ani însă, prin apariţia microprocesoarelor Intel (şi a produselor clonă) ce au preluat multe dintre conceptele tehnologiei clasice RISC, diferenţa dintre performanţele celor două tipuri arhitecturale s-a micşorat vizibil, ajungându-se ca cele mai noi procesoare Pentium III şi Pentium 4 să concureze cu succes procesoarele RISC.

Se presupune că nu se va renunţa foarte uşor în viitor la vechea arhitectură CISC (care are însă are şi va prelua în continuare dintre beneficiile RISC), ajungându-se poate la situaţia în care vom putea cu greu să spunem diferenţele dintre cele două tipuri arhitecturale ce erau atât de diferite în trecut. Următoarele microprocesoare x86, precum cele bazate pe nucleele *Mustang şi Sledgehammer* de la AMD sau *Pentium 4, Foster şi Itanium* (ultimul pe 64 de biţi) de la Intel vor reduce din ce în ce mai mult "gaura" ce desparte cele două variante tehnologice.

# 3

# INTRODUCERE ÎN LIMBAJUL DE ASAMBLARE INTEL

## 3.1 Elementele arhitecturale de bază ale microprocesorului Intel

Ne vom referi în cele ce urmează la familia de microprocesoare intitulată Intel iAPx86 ce stau la baza calculatoarelor IBM PC, începând de la procesoarele 8088 şi 8086, continuând cu 80286, 80386, 80486, Pentium, ş.a.m.d. Procesorul 8086 reprezintă, de fapt, baza familiei ce este cunoscută pe scurt sub denumirea de familia microprocesoarelor x86. De aceea se vor face referiri în continuare la această arhitectură (8086).

### *3.1.1 Regiştrii microprocesorului Intel*

Regiştrii (sau registrele) microprocesorului reprezintă locaţii de memorie speciale aflate direct pe cip; din această cauză reprezintă cel mai rapid tip de memorie. Alt lucru deosebit legat de regiştri este faptul că fiecare dintre aceştia au un scop bine precizat, oferind anumite funcţionalităţi speciale, unice. Există patru mari categorii de regiştri: regiştrii de uz general, registrul indicatorilor de stare (*flags*), regiştrii de segment şi registrul pointer de instrucţiune.

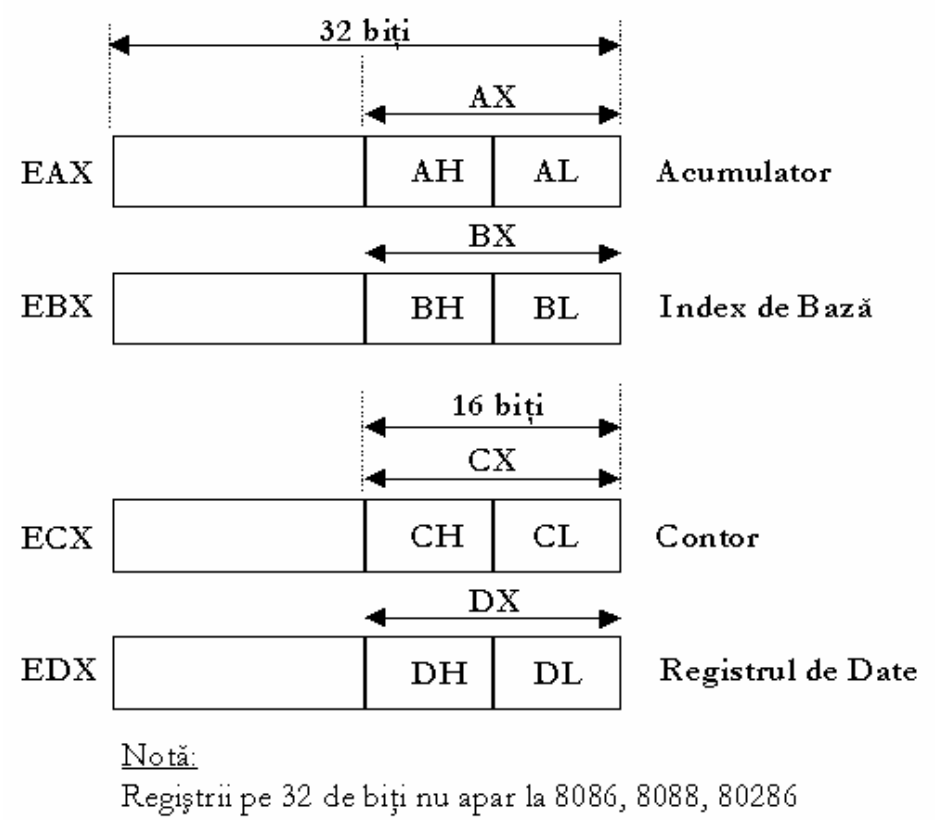



Fig. 3.1 Regiştrii de uz general – acumulator, index de bază, contor şi de date

### *3.1.1.1 Regiştrii de uz general*

Regiştrii de uz general (figura 3.1 şi figura 3.2) sunt implicaţi în operarea majorităţii instrucţiunilor, drept operanzi sursă sau destinaţie pentru calcule, copieri de date, pointeri la locaţii de memorie sau cu rol de contorizare. Fiecare dintre cei 8 regiştrii de uz general AX, BX, CX, DX, SP, BP, DI,SI sunt regiştrii pe 16 biţi pentru microprocesorul 8086, iar de la procesorul 80386 încoace au devenit regiştrii pe 32 de biţi, denumiţi, respectiv: EAX, EBX, ECX, EDX, ESP, EBP, EDI, ESI (litera E provine de la *Extended – extins* în engleză). Mai mult, cei mai puţin semnificativi 8 biţi ai regiştrilor AX, BX, CX, DX formează respectiv regiştrii AL, BL, CL, DL (litera L provine de la *Low – jos* în engleză), iar cei mai semnificativi 8 biţi

ai aceloraşi regiştrii formează regiştrii AH, BH, CH, DH (litera H provine de la *High – înalt* în engleză) (figura 3.1).

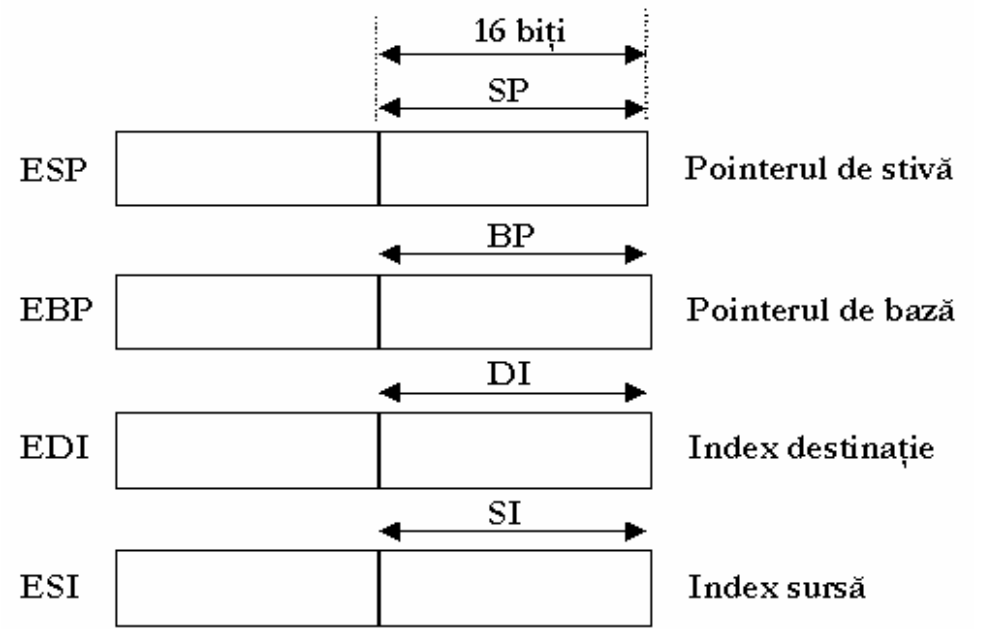



Fig. 3.2 Regiştrii de uz general index şi pointer

Ne vom concentra în continuare atenția asupra regiştrilor generali pe 16 biți; fiecare dintre aceştia poate stoca o valoare pe 16 biți, poate fi folosit pentru stocarea unei valori din memorie sau poate fi utilizat pentru operații aritmetice şi logice. Spre exemplu, următoarele instrucțiuni:

```
…
MOV   BX, 2
MOV  DX, 3
ADD  BX, DX
…
```

Încarcă valoarea 2 în registrul BX, valoarea 3 în registrul DX, adună cele două valori iar rezultatul (5) este memorat în registrul BX. În exemplul anterior putem utiliza oricare dintre regiştrii de uz general în locul regiştrilor BX şi DX. În afara proprietății de  a stoca valori şi de a folosi drept operanzi sursă sau destinație pentru instrucțiunile de manipulare a datelor, fiecare

dintre cei opt regiştri de uz general au propria "personalitate". Vom vedea în continuare care sunt caracteristicile specifice fiecăruia dintre regiştrii de uz general.

**Registrul AX (EAX)**

Registrul AX (EAX) este denumit şi registrul *acumulator*, fiind principalul registru de uz general utilizat pentru operaţii aritmetice, logice şi de deplasare de date. Totdeauna operaţiile de înmulţire şi împărţire presupun implicarea registrului AX. Unele dintre instrucţiuni sunt optimizate pentru a se executa mai rapid atunci când este folosit AX. În plus, registrul AX este folosit şi pentru toate transferurile de date de la/către porturile de Intrare/Ieşire. Poate fi accesat pe porţiuni de 8, 16 sau 32 de biţi, fiind referit drept AL (cei mai puţin semnificativi 8 biţi din AX), AH (cei mai semnificativi 8 biţi din AX), AX (16 biţi) sau EAX (32 de biţi). Prezentăm în continuare alte câteva exemple de instrucţiuni ce utilizează registrul AX. De remarcat este faptul că transferurile de date se fac pentru instrucţiunile (denumite şi *mnemonice*) Intel de la dreapta spre stânga, exact invers decât la Motorola (vom vedea şi alt exemplu asemănător la scrierea datelor în memorie sub format diferit la Motorola faţă de Intel), unde transferul se face de la stânga la dreapta.

Instrucţiunea: **MOV AX, 1234H** încarcă valoarea 1234H (4660 în zecimal) în registrul acumulator AX. După cum spuneam, cei mai puţini semnificativi 8 biţi ai registrului AX sunt identificaţi de AL (A-Low) iar cei mai semnificativi 8 biţi ai aceluiaşi registru sunt identificaţi ca fiind AH (A-High). Acest lucru este utilizat pentru a lucra cu date pe un octet, permiţând ca registrul AX să fie folosit pe postul a doi regiştri separaţi (AH şi AL). Aceeaşi regulă este valabilă şi pentru regiştrii de uz general BX, CX, DX. Următoarele trei instrucţiuni setează registrul AH cu valoarea 1, incrementează cu 1 această valoare şi apoi o copiază în registrul AL:

```
…
MOV  AH, 1
INC  AH
MOV AL, AH
…
```

Valoarea finală a registrului AX va fi 22 (AH = AL = 2).

**Registrul BX (EBX)**

Registrul BX (Base), sau *registrul de bază* poate stoca adrese pentru a face referire la diverse structuri de date, cum ar fi vectorii stocați în memorie. O valoare reprezentată pe 16 biți stocată în registrul BX poate fi utilizată ca fiind o porțiune din adresa unei locații de memorie ce va fi accesată. Spre exemplu, următoarele instrucțiuni încarcă registrul AH cu valoarea din memorie de la adresa 21.

```
…
MOV  AX,  0
MOV  DS, AX
MOV  BX, 21
MOV  AH, [ BX ]
…
```

Se observă că am încărcat valoarea 0 în registrul DS înainte de a accesa locația de memorie referită de registrul BX. Acest lucru este datorat segmentării memoriei (segmentare discutată mai în detaliu în secțiunea consacrată regiştrilor de segment); implicit, atunci când este folosit ca pointer de memorie, BX face referire relativă la registrul de segment DS (adresa la care face referire este o adresă relativă la adresa de segment conținută în registrul DS).

**Registrul CX (ECX)**

Specializarea registrului CX (Counter) este numărarea; de aceea, el se numeşte şi registrul contor. De asemenea, registrul CX joacă un rol special atunci când se foloseşte instrucțiunea LOOP. Rolul de contor al registrului CX se observă imediat din exemplul următor:

```
…
MOV CX, 5
start:
…
<instrucțiuni ce se vor executa de 5 ori>
…
SUB CX, 1
```

```
JNZ start
…
```

Deoarece valoarea inițială a lui CX este 5, instrucțiunile cuprinse între eticheta start şi instrucțiunea JNZ se vor executa de 5 ori (până când registrul CX devine 0). Instrucțiunea SUB CX, 1 decrementează registrul CX cu valoarea 1 iar instrucțiunea JNZ start determină saltul înapoi la eticheta start dacă CX nu are valoarea 0. În limbajul microprocesorului există şi o instrucțiune specială legată de ciclare. Aceasta este instrucțiunea LOOP, care este folosită în combinație cu registrul CX. Liniile de cod următoare sunt echivalente cu cele anterioare, dar aici se utilizează instrucțiunea LOOP:

```
…
MOV CX, 5
start:
…
<instrucțiuni ce se vor executa de 5 ori>
…
LOOP start
…
```

Se observă că instrucțiunea LOOP este folosită în locul celor două instrucțiuni SUB şi JNZ anterioare; LOOP decrementează automat registrul CX cu 1 şi execută saltul la eticheta specificată (*start*) dacă CX este diferit de zero, totul într-o singură instrucțiune.

**Registrul DX (EDX)**

Registrul de uz general DX (Data register), denumit şi registrul de date, poate fi folosit în cazul transferurilor de date Intrare/Ieşire sau atunci când are loc o operație de înmulțire sau de împărțire. Instrucțiunea **IN AL, DX** copiază o valoare de tip Byte dintr-un port de intrare, a cărui adresă se află în registrul DX. Următoarele instrucțiuni determină scrierea valorii 101 în portul I/O 1002:

```
...
MOV  AL, 101
```

```
MOV  DX, 1002
OUT  DX, AL
…
```

Referitor la operațiile de înmulțire şi împărțire, atunci când împărțim un număr pe 32 de biți la un număr pe 16 biți, cei mai semnificativi 16 biți ai deîmpărțitului trebuie să fie în DX. După împărțire, restul împărțirii se va afla în DX. Cei mai puțin semnificativi 16 biți ai deîmpărțitului trebuie să fie în AX iar câtul împărțirii va fi în AX. La înmulțire, atunci când se înmulțesc două numere pe 16 biți, cei mai semnificativi 16 biți ai produsului vor fi stocați în DX iar cei mai puțin semnificativi 16 biți în registrul AX.

**Registrul SI**

Registrul SI (Source Index) poate fi folosit, ca şi BX, pentru a referi adrese de memorie. De exemplu, secvența de instrucțiuni următoare:

```
…
MOV  AX, 0
MOV  DS, AX
MOV  SI, 33
MOV  AL, [ SI ]
```

Încarcă valoarea (pe 8 biți) din memorie de la adresa 33 în registrul AL. Registrul SI este, de asemenea, foarte folositor atunci când este utilizat în legătură cu instrucțiunile dedicate tipului STRING (şir de caractere). Secvența următoare :

```
...
CLD
MOV  AX, 0
MOV  DS, AX
MOV  SI, 33
LODSB
...
```

nu numai că încarcă registrul AX cu valoarea de la adresa de memorie referită de registrul SI, dar adună, de asemenea, valoarea 1 la SI. Acest lucru este deosebit de eficient atunci când se accesează secvenţial o serie de locaţii de memorie, cum ar fi şirurile de caractere. Instrucţiunile de tip STRING se pot repeta de mai multe ori, astfel încât o singură instrucţiune poate avea ca efect sute sau mii de operaţii.

**Registrul DI**

Registrul DI (Destination Index) este utilizat în mod asemănător registrului SI. În secvenţa de instrucţiuni următoare:

```
...
MOV  AX, 0
MOV  DS, AX
MOV  DI, 1000
ADD  BL, [ DI ]
…
```

se adună la registrul BL valoarea pe 8 biţi stocată la adresa 1000. Registrul DI este puţin diferit faţă de registrul SI în cazul instrucţiunilor de tip string; dacă SI este întotdeauna pe post de pointer sursă de memorie, registrul DI serveşte drept pointer destinaţie de memorie. Mai mult, în cazul instrucţiunilor de tip string, registrul SI adresează memoria relativ la registrul de segment DS, în timp ce DI conţine referiri la memorie relativ la registrul de segment ES. În cazul în care SI şi DI sunt utilizaţi cu alte instrucţiuni, ei fac referire la registrul de segment DS.

**Registrul BP**

Pentru a înţelege mai bine rolul regiştrilor BP şi SP, a sosit momentul să spunem câteva lucruri despre porţiunea de memorie denumită stivă (în engleză *stack*). Stiva (figura 3.3) reprezintă o porţiune specială de locaţii adiacente din memorie. Aceasta este conţinută în cadrul unui segment de memorie şi identificată de un selector de segment memorat în registrul SS (cu excepţia cazului în care se foloseşte modelul nesegmentat de memorie în care stiva poate fi localizată oriunde în spaţiul de adrese liniare al programului). Stiva este o porţiune a memoriei unde valorile pot fi stocate

şi accesate pe principul LIFO (Last In – First Out), drept urmare ultima valoare stocată în stivă este prima ce va fi citită din stivă. De regulă, stiva este utilizată la apelul unei proceduri sau la întoarcerea dintr-un apel de procedură (principalele instrucţiuni folosite sunt CALL şi RET).

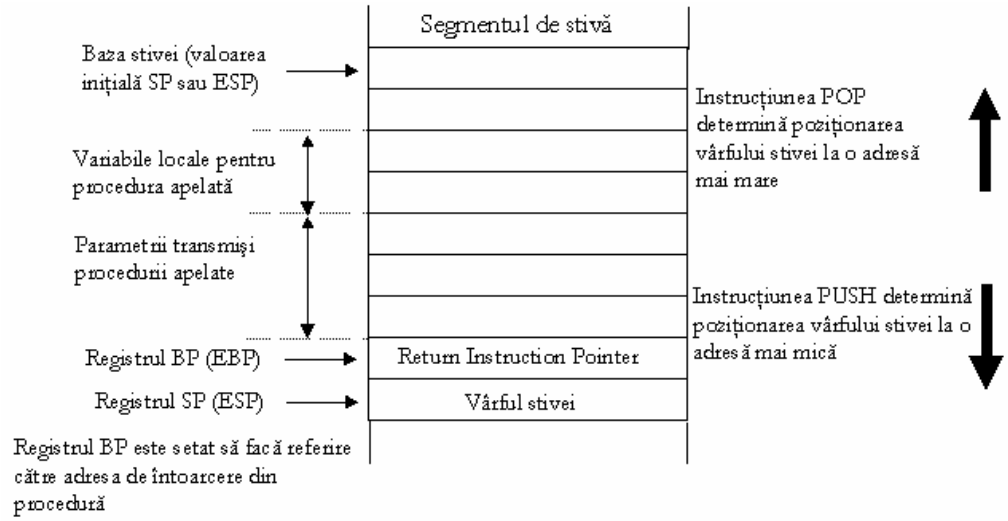

Segmentul de stivă
Baza stivei (valoarea inițială SP sau ESP)
Instrucțiunea POP determină poziționarea vârfului stivei la o adresă mai mare
Variabile locale pentru procedura apelată
Parametrii transmiși procedurii apelate
Instrucțiunea PUSH determină poziționarea vârfului stivei la o adresă mai mică
Registrul BP (EBP)
Return Instruction Pointer
Registrul SP (ESP)
Vârful stivei
Registrul BP este setat să facă referire către adresa de întoarcere din procedură

Fig. 3.3 Structura stivei

Registrul pointer de bază, BP (Base Pointer) poate fi utilizat ca pointer de memorie precum regiştrii BX, SI şi DI. Diferenţa este aceea că, dacă BX, SI şi DI sunt utilizaţi în mod normal ca pointeri de memorie relativ la segmentul DS, registrul BP face referire relativ la segmentul de stivă SS. Principiul este următorul: o modalitate de a trece parametrii unei subrutine este aceea de a utiliza stiva (acest lucru se întâmplă în mod obişnuit în limbajele de nivel înalt, C sau Pascal, spre exemplu). Dacă stiva se află în porţiunea de memorie referită de registrul de segment SS (Stack Segment), datele se află în mod normal în segmentul de memorie referit de către DS, registrul segment de date. Deoarece BX, SI şi DI se referă la segmentul de date, nu există o modalitate eficientă de a folosi regiştrii BX, SI, DI pentru a face referire la parametrii salvaţi în stivă din cauză că stiva este localizată într-un alt segment de memorie. Registrul BP oferă

rezolvarea acestei probleme asigurând adresarea în segmentul de stivă. Spre exemplu, instrucţiunile:

```
...
PUSH  BP
MOV  BP, SP
MOV  AX, [ BP+4 ]
…
```

fac să se acceseze segmentul de stivă pentru a încărca registrul AX cu primul parametru trimis de un apel C unei rutine scrise în limbaj de asamblare. În concluzie, registrul BP este conceput astfel încât să ofere suport pentru accesul la parametri, variabile locale şi alte necesităţi legate de accesul la porţiunea de stivă din memorie.

**Registrul SP**

Registrul SP (Stack Pointer), sau pointerul de stivă, reţine de regulă adresa de deplasament a următorului element disponibil în cadrul segmentului de stivă. Acest registru este, probabil, cel mai puţin „general" dintre regiştrii de uz general, deoarece este dedicat mai tot timpul administrării stivei.

Registrul BP face în fiecare clipă referire la vârful stivei – acest vârf al stivei reprezintă adresa locaţiei de memorie în care va fi introdus următorul element în stivă. Acţiunea de a introduce un nou element în stivă se numeşte „împingere" (în engleză *push*); de aceea, instrucţiunea respectivă poartă numele de PUSH. În mod asemănător, operaţia de scoatere a unui element din vârful stivei poartă, în engleză, numele de *pop*, iar instrucţiunea echivalentă operaţiei se numeşte POP. În figurile 3.4 şi 3.5 sunt ilustrate modificările survenite în conţinutul stivei şi al regiştrilor SP, BX şi CX ca urmare a execuţiei instrucţiunilor următoare (se presupune că registrul SP are iniţial valoarea 1000):

```
...
MOV  BX, 9
PUSH  BX
MOV  CX, 10
PUSH  CX
```

```
POP  BX
POP  CX
...
```

Este permisă stocarea valorilor în registrul SP precum şi modificarea valorii sale prin adunare sau scădere la fel ca şi în cazul celorlalți regiştri de uz general; totuşi, acest lucru nu este recomandat dacă nu suntem foarte siguri de ceea ce facem. Prin modificarea registrului SP, vom modifica adresa de memorie a vârfului stivei, ceea ce poate avea efecte neprevăzute, aceasta pentru că instrucțiunile PUSH şi POP nu reprezintă unicele modalități de utilizare a stivei. Indiferent dacă apelăm o subrutină sau ne întoarcem dintr-un astfel de apel de subrutină, fie procedură sau funcție, în acest caz este folosită stiva. Unele resurse de sistem, precum tastatura sau ceasul de sistem, pot folosi stiva în momentul trimiterii unei întreruperi la microprocesor. Acest lucru presupune că stiva este folosită continuu, deci dacă se modifică registrul SP (adică adresa stivei), datele din noile locații de memorie nu vor mai fi cele corecte. În concluzie, registrul SP nu trebuie modificat în mod direct; el este modificat automat în urma instrucțiunilor POP, PUSH, CALL, RET. Oricare dintre ceilalți regiştri de uz general pot fi modificați în mod direct în orice moment.

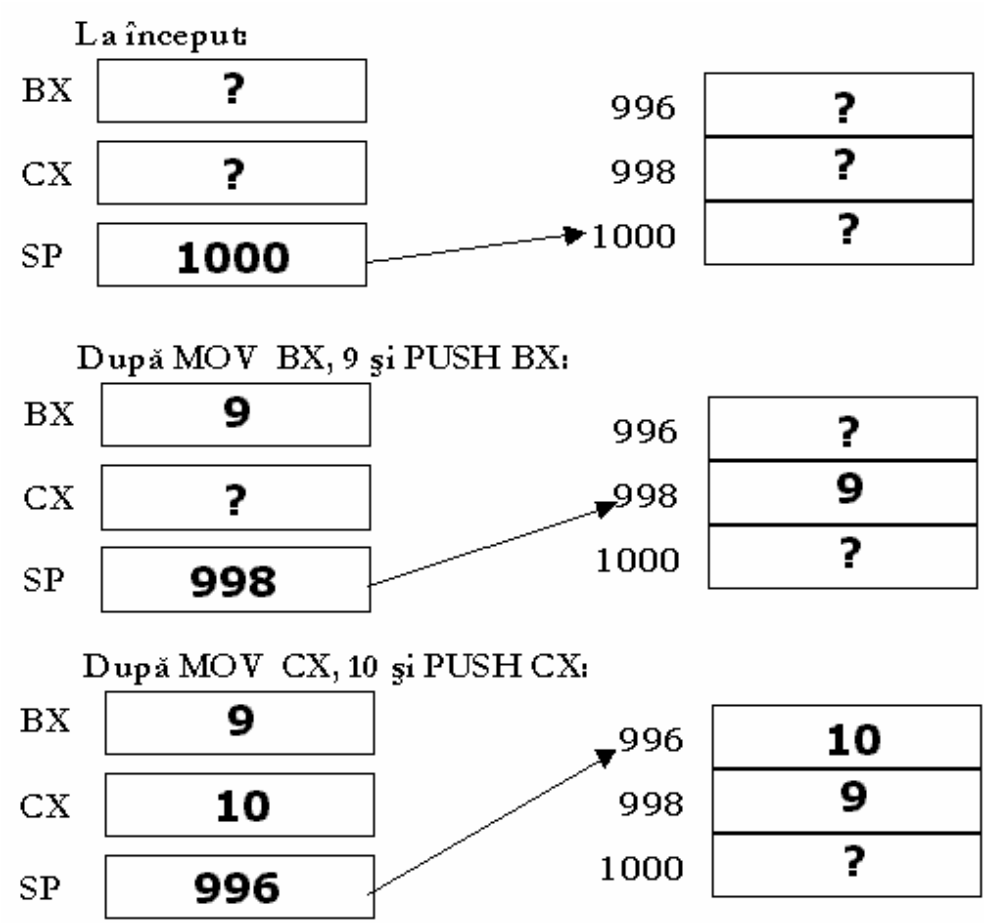



Fig. 3.4 Modalitatea de funcționare a stivei după execuția primelor patru instrucțiuni

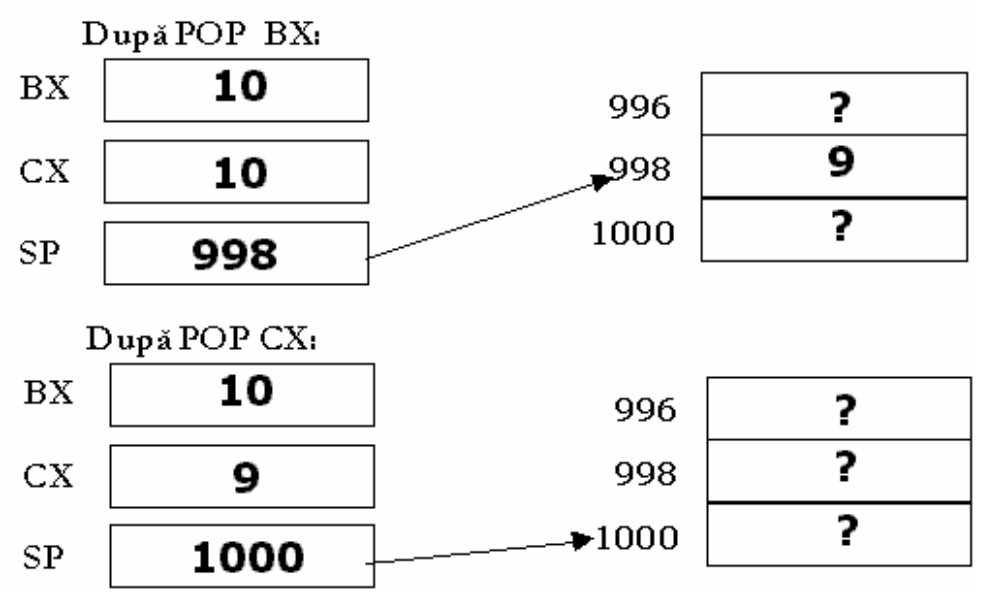



Fig. 3.5 Funcționarea stivei după ultimile două instrucțiuni POP

### *3.1.1.2 Registrul pointer de instrucțiuni (IP)*

Registrul pointer de instrucțiuni (IP – Instruction Pointer, vezi figura 3.6) este folosit, întotdeauna, pentru a stoca adresa următoarei instrucțiuni ce va fi executată de către microprocesor. Pe măsură ce o instrucțiune este executată, pointerul de instrucțiune este incrementat şi se va referi la următoarea adresă de memorie (unde este stocată următoarea instrucțiune ce va fi executată). De regulă, instrucțiunea ce urmează a fi executată se află la adresa imediat următoare instrucțiunii ce a fost executată, dar există şi cazuri speciale (rezultate fie din apelul unei subrutine prin instrucțiunea CALL, fie prin întoarcerea dintr-o subrutină, prin instrucțiunea RET). Pointerul de instrucțiuni nu poate fi modificat sau citit în mod direct; doar instrucțiuni speciale pot încărca acest registru cu o nouă valoare. Registrul pointer de instrucțiune nu specifică pe de-a întregul adresa din memorie a următoarei instrucțiuni ce va fi executată, din aceeaşi cauză a segmentării memoriei. Pentru a aduce o instrucțiune din memorie, registrul CS oferă o adresă de bază iar registrul pointer de instrucțiune indică adresa de deplasament plecând de la această adresă de bază.

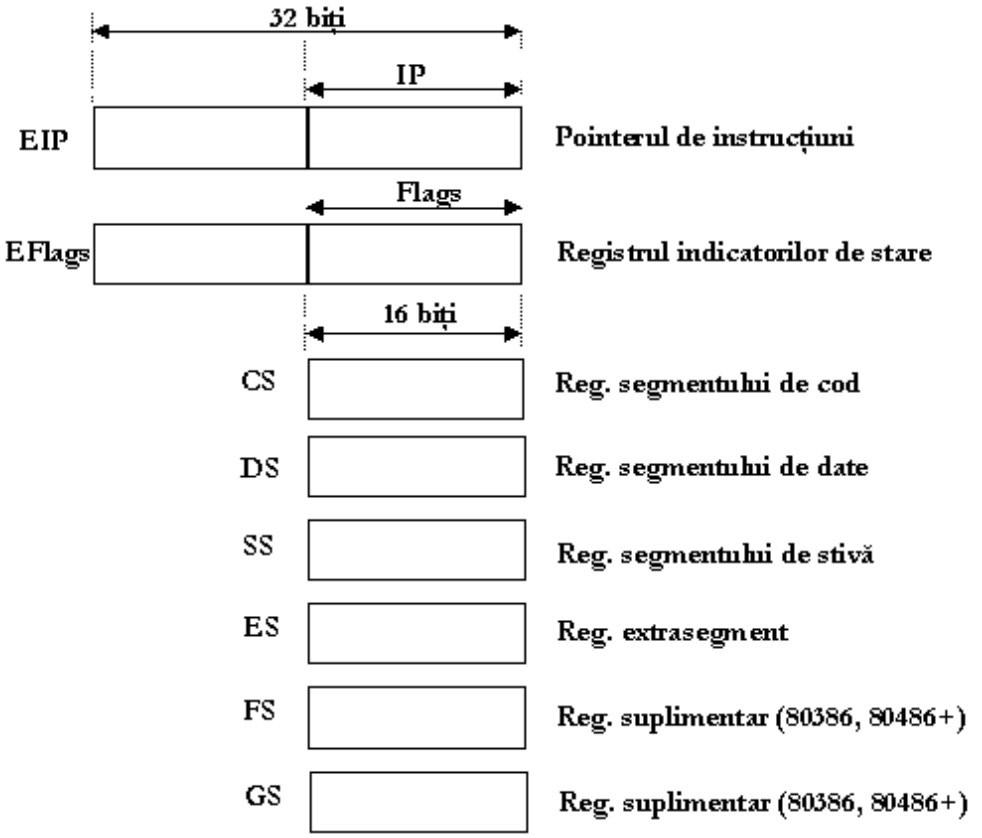



Fig. 3.6. Regiştrii de segment, pointerul de instrucțiuni şi registrul indicatorilor de stare

### *3.1.1.3 Registrul indicatorilor de stare (FLAGS)*

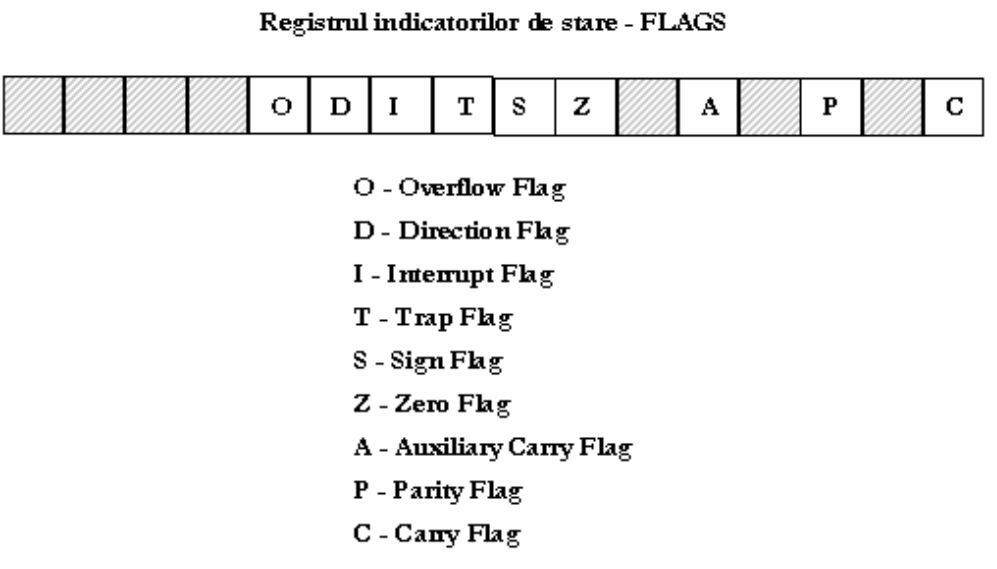



Fig. 3.7 Registrul indicatorilor de stare - detaliu

Registrul indicatorilor de stare (FLAGS) pe 16 biți conține informații legate de starea microprocesorului precum și de rezultatele ultimilor instrucțiuni executate. Un indicator de stare (*flag*) este în sine o locație de memorie de 1 bit ce indică starea curentă a microprocesorului și modalitatea sa de operare. Un indicator se spune că "este setat" dacă are valoarea 1 și "nu este setat" în caz contrar. Indicatorii de stare se modifică după execuția unor instrucțiuni aritmetice sau logice. Exemple de indicatori de stare (vezi figura 3.7):

- C (Carry) indică apariția unei cifre binare de transport în cazul unei adunări sau împrumut în cazul unei scăderi;
- O (Overflow) apare în urma unei operații aritmetice. Dacă este setat, înseamnă că rezultatul nu încape în operandul destinație;
- Z (Zero) indică faptul că rezultatul unei operații aritmetice sau logice este zero;
- S (Sign) indică semnul rezultatului unei operații aritmetice;

- D (Direction) – când este zero, procesarea elementelor şirului se face de la adresa mai mică la cea mai mare, în caz contrar este invers;
- I (Interrupt) controlează posibilitatea microprocesorului de a răspunde la evenimente externe (apeluri de întrerupere);
- T (Trap) este folosit de programele de depanare (de tip debugger), activând sau nu posibilitatea execuției programului pas cu pas. Dacă este setat, UCP întrerupe fiecare instrucțiune, lăsând programul depanator să execute programul respectiv pas cu pas;
- A (Auxiliary carry) suportă operații în codul BCD. Majoritatea programelor nu oferă suport pentru reprezentarea numerelor în acest format, de aceea se utilizează foarte rar;
- P (Parity) este setat în conformitate cu paritatea biților cei mai puțin semnificativi ai unei operații cu date. Astfel, dacă rezultatul unei operații conține un număr par de biți 1, acest indicator este setat. Dacă numărul de biți 1 din rezultat este impar, atunci indicatorul PF este zero. Este folosit de regulă de programe de comunicații, dar Intel a introdus acest indicator nu pentru a îndeplini o anumită funcționalitate, ci pentru a asigura compatibilitatea cu vechile microprocesoare ale familiei x86.

#### *3.1.1.4 Regiştrii de segment*

Proprietățile regiștrilor de segment (figura 3.6) sunt în strânsă legătură cu noțiunea de segmentare a memoriei. Premisa de la care se pleacă este următoarea: 8086 este capabil să adreseze 1MB de memorie, astfel că sunt necesare adrese pe 20 de biți pentru a cuprinde toate locațiile din spațiul de 1 MB de memorie. Totuşi, registrele utilizate sunt registre pe 16 biți, deci a trebuit să se găsească o soluție pentru această problemă. Soluția găsită se numeşte *segmentarea memoriei*; în acest caz memoria de 1MB este împărțită în 16 segmente de câte 64 KB (16*64 KB = 1024 KB = 1 MB).

Noțiunea de segmentare a memoriei presupune utilizarea unor adrese de memorie formate din două părți. Prima parte reprezintă adresa segmentului iar cea de-a doua porțiune reprezintă adresa de deplasament, sau offset-ul (figura 3.8).

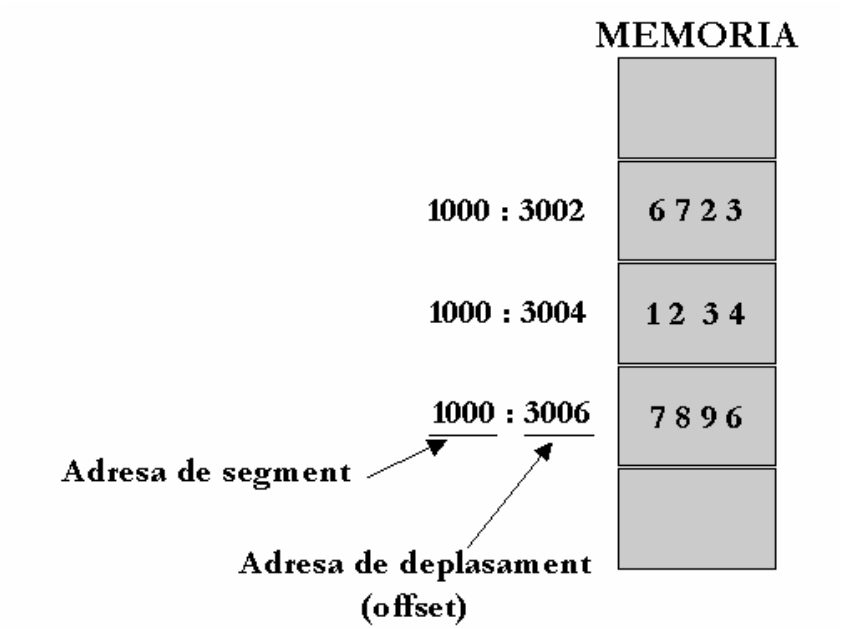



Fig. 3.8 Cele două porțiuni ale unei adrese segmentate

Fiecare pointer de memorie pe 16 biți este combinat cu conținutul unui registru de segment pe 16 biți pentru a forma o adresă completă pe 20 de biți. Adresa de segment împreună cu adresa de deplasament sunt combinate în felul următor: valoarea de segment este deplasată la stânga cu 4 biți (înmulțită cu 16 = $2^4$) şi apoi adunată cu valoarea adresei de deplasament. Adresa astfel construită se numeşte adresă efectivă; fiind o adresă pe 20 de biți poate accesa $2^{20}$ octeți de memorie, adică 1 MB de memorie. Construirea adresei efective este prezentată în figura 3.9.

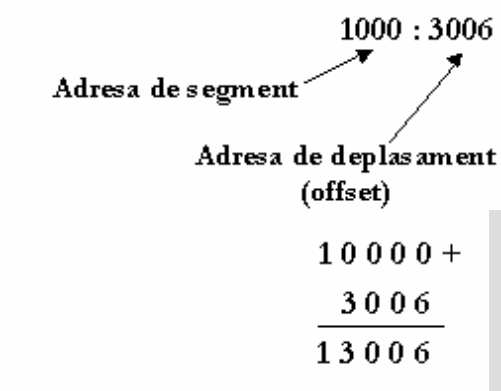

1000 : 3006
Adresa de segment
Adresa de deplasament
(offset)
10000 +
3006
13006
- adresa de segment se deplasează la stânga cu 4 biți – o cifră hexa
- se adună adresa de deplasament
- se obține adresa efectivă pe 20 de biți (5 cifre hexa)

Fig. 3.9 Exemplu de calcul al adresei efective

Registrul CS – acest registru face referire la începutul blocului de 64 KB de memorie în care se află codul programului (segmentul de cod). Microprocesorul 8086 nu poate aduce altă instrucțiune pentru execuție decât cea definită de CS. Registrul CS poate fi modificat de un număr de instrucțiuni, precum instrucțiuni de salt, apel sau de întoarcere. El nu poate fi încărcat în mod direct cu o valoare, ci doar prin intermediul unui alt registru general.

Registrul DS – face referire către începutul segmentului de date, unde se află mulțimea de date cu care lucrează programul aflat în execuție.

Registrul ES – face referire la începutul blocului de 64 KB cunoscut sun denumirea de extrasegment. Acesta nu este dedicat nici unui scop anume, fiind disponibil pentru diverse acțiuni. Uneori acesta poate fi folosit pentru creearea unui bloc de memorie de 64 KB adițional pentru date. Acest extrasegment lucrează foarte bine în cazul instrucțiunilor de tip STRING. Toate instrucțiunile de tip STRING ce scriu în memorie folosesc adresarea ES : DI ca adresă de memorie.

Registrul SS – face referire la începutul segmentului de stivă, care este blocul de 64 KB unde se află stiva. Toate instrucțiunile ce folosesc implicit registrul SP (instrucțiunile POP, PUSH, CALL, RET) lucrează în segmentul de stivă deoarece registrul SP este capabil să adreseze memoria doar în segmentul de stivă.

## 3.2 Elemente ale limbajului de asamblare

### *3.2.1 Formatul general al unei instrucțiuni în limbaj de asamblare*

O linie de cod scrisă în limbaj de asamblare are următorul format general:

**<nume> <instrucțiune/directivă> <operanzi> <;comentariu>**

unde:

- **<nume>** - reprezintă un nume simbolic opțional;
- **<instrucțiune/directivă>** - reprezintă mnemonica (numele) unei instrucțiuni sau a unei directive;
- **<operanzi>** - reprezintă o combinație de unul, doi sau mai mulți operanzi (sau chiar nici unul), care pot fi constante, referințe de memorie, referințe de regiştri, şiruri de caractere, în funcție de structura particulară a instrucțiunii;
- **<;comentariu>** - reprezintă un comentariu opțional ce poate fi plasat după caracterul „;” până la sfârşitul liniei respective de cod.

### *3.2.2 Nume de variabile şi etichete*

Numele folosite într-un program scris în limbaj de asamblare pot identifica variabile numerice, variabile şir de caractere, locații de memorie sau etichete. Spre exemplu, următoarea secvență de cod, care calculează valoarea lui trei factorial (3! = 1 x 2 x 3 = 6) cuprinde câteva nume de variabile şi etichete:

```
.MODEL  small
.STACK  200h
.DATA
Valoare_Factorial  DW  ?
Factorial  DW  ?
```

```
.CODE

Trei_Factorial  PROC
MOV  ax, @data
MOV ds, ax
MOV  [Valoare_Factorial], 1
MOV [Factorial], 2
MOV  cx, 2
Ciclare:
MOV ax, [Valoare_Factorial]
MUL [Factorial]
MOV  [Valoare_Factorial], ax
INC [Factorial]
LOOP Ciclare
RET
Trei_Factorial ENDP
END
```

Numele *Valoare_Factorial* şi *Factorial* sunt utilizate pentru definirea a două variabile de tip word (pe 16 biți), *Trei_Factorial* identifică numele procedurii (subrutinei) ce conține codul pentru calculul factorialului, permițând apelul său din altă parte a programului. *Ciclare* reprezintă un nume de etichetă, identificând adresa instrucțiunii MOV ax, [Valoare_Factorial], astfel încât instrucțiunea LOOP folosită mai jos să poată face un salt înapoi la această instrucțiune. Numele de variabile pot conține următoarele caractere: literele a-z şi A-Z, cifrele de la 0-9 precum şi caracterele speciale _ (*underscore* – liniuță de subliniere), @ (*„at”* în engleză – citit şi *„a rond”* sau *„coadă de maimuță”*), $ şi ?. Se poate folosi si caracterul punct (“.”) drept prim caracter al numelui unei etichete. Cifrele 0-9 nu pot fi utilizate pe prima poziție a numelui; de asemenea, nu pot fi folosite nume care să conțină un singur caracter $ sau ?. Fiecare nume poate fi definit *o singură dată* (numele sunt *unice*) şi pot fi utilizate ca operanzi de oricâte ori se doreşte într-un program. Un nume poate să apară într-un

program singur pe o linie (linia respectivă nu mai conține altă instrucțiune sau directivă). În acest caz, valoarea numelui este dată de adresa instrucțiunii sau directivei de pe linia următoare din program. De exemplu, în secvența următoare:

```
...
JMP scădere
...
scădere:
SUB AX, CX
...
```

următoarea instrucțiune care va fi executată după instrucțiunea **JMP scadere** va fi instrucțiunea **SUB AX, CX**. Exemplul anterior este echivalent cu secvența:

```
...
JMP scădere
...
scădere: SUB AX, CX
...
```

Există unele avantaje atunci când scriem instrucțiunile pe linii separate. În primul rând, atunci când scriem un nume de etichetă pe o singură linie, este mai uşor să folosim nume lungi de etichete fără a strica „forma" programului scris în limbaj de asamblare. În al doilea rând, este mai uşor să adăugăm ulterior o nouă instrucțiune în dreptul etichetei dacă aceasta nu este scrisă pe aceeaşi linie cu instrucțiunea.

Numele variabilelor sau etichetelor folosite într-un program nu trebuie să se confunde cu numele *rezervate* de asamblor, cum ar fi numele de directive şi instrucțiuni, numele regiştrilor, etc. De exemplu, o declarație de genul:

```
...
ax  DW  0
BYTE:
...
```

nu poate fi acceptată, deoarece AX este numele registrului acumulator, AX, iar BYTE reprezintă un cuvânt cheie rezervat.

Orice nume de etichetă ce apare pe o linie fără instrucțiuni sau apare pe o linie cu instrucțiuni trebuie să aibă semnul „:” după numele ei. Totodată, se încearcă să se dea un nume sugestiv etichetelor din program.

Fie următorul exemplu:

```
...
CMP AL, ‘a’
JB  Nu_este_literă_mică
CMP AL, ‘z’
JA  Nu_este_literă_mică
SUB AL, 20H          ; se transforma in litera mare
Nu_este_literă_mică:
…
```

comparativ cu:

```
...
CMP AL, ‘a’
JB  x5
CMP AL, ‘z’
JA  x5
SUB AL, 20H          ; se transformă în literă mare
x5:
…
```

Dacă în primul caz am folosit un nume sugestiv de etichetă (Nu_este_literă_mică), în cazul al doilea, identic din punct de vedere al funcționalității cu primul, eticheta a fost denumită x5, absolut nesugestiv!

***Observație:***

Limbajul de asamblare nu este *case sensitive*. Aceasta semnifică faptul că, într-un program scris în limbaj de asamblare, numele de variabile, etichete, instrucțiuni, directive, mnemonice etc., pot fi scrise atât cu litere mari cât şi cu litere mici, nefăcându-se diferența între ele (*Nu _ este _ literă _ mică* este acelaşi lucru cu *nu_este_literă_mică* sau *Nu_Este_Literă_Mică* etc.).

### *3.2.3 Directive de segment simplificate*

Datorită faptului că regiştrii microprocesorului 8086 sunt regiştrii pe 16 biţi, s-a impus folosirea unor segmente de memorie de câte 64 Ko (maxim cât se poate adresa având la dispoziţie 16 biţi – 64 Ko = 2 ^ 16 = 65536). Într-un program scris în limbaj de asamblare (vom folosi în continuare prescurtarea ASM) există trei segmente: segmentul de cod, segmentul de date şi segmentul de stivă.

Directivele de segment (fie sub formă standard, fie sub formă simplificată) sunt necesare în orice program scris în limbaj de asamblare pentru a defini şi controla utilizarea segmentelor iar directiva END este folosită întotdeauna pentru a încheia codul programului.

Exemple de directive de segment simplificate sunt:

```
.STACK
.CODE
.DATA
.MODEL
DOSSEG
END
```

.STACK,.CODE,.DATA definesc, respectiv, segmentele de stivă, de cod şi de date.

De exemplu, **.STACK 200H** defineşte o stivă de 512 octeţi (în ASM valorile ce sunt încheiate cu litera H semnifică faptul că este vorba despre hexazecimal). O astfel de valoare pentru stivă este suficientă în mod normal; unele programe, însă (îndeosebi cele recursive) pot necesita dimensiuni mai mari ale stivei.

Directiva **.CODE** marchează începutul segmentului de cod.

Directiva **.DATA** marchează începutul segmentului de date, adică locul în care vom plasa variabilele de memorie. Reprezentativ aici este faptul că trebuie încărcat în mod explicit registrul de segment DS cu valoarea "@data" înaintea accesării locaţiilor de memorie în segmentul definit de .DATA. Având în vedere că un registru de segment poate fi încărcat fie dintr-un registru general fie dintr-o locaţie de memorie dar nu

poate fi încărcat direct cu o constantă, registrul de segment DS este încărcat în general printr-o secvență de două instrucțiuni:

```
...
mov ax, @data
mov ds, ax
...
```

(se poate folosi şi alt registru general în locul lui AX).

Secvența anterioară semnifică faptul că DS se va referi către segmentul de date ce începe cu directiva .DATA.

Considerăm în continuare un exemplu de program ce afişează textul memorat în *DataString* pe ecran:

```
;Program p01.asm
.MODEL small                ;se specifică modelul de memorie
                            ;SMALL
.STACK 200H                 ;se defineşte o stivă de 512 octeți
.DATA                       ;se specifică începutul segmentului de
                            ;date
DataString DB 'Hello!$'     ;se defineşte variabila
                            ;DataString, inițializată cu valoarea
                            ;"Hello!"
.CODE                       ;începutul segmentului de cod al
                            ;programului
ProgramStart:               ;orice program are o etichetă de
                            ;început
mov bx,@data                ;secvența ce setează registrul DS să
                            ;facă referire la segmentul de date ce
                            ;începe cu .DATA
mov ds,bx
mov dx, OFFSET DataString ;se încarcă în DX adresa
                            ;variabilei DataString
mov ah,09                   ;codul funcției DOS de afişare a unui
                            ;string
```

```
int 21H                 ;apelul DOS de afişare a string-ului
mov ah, 4cH             ;codul funcţiei DOS de terminare a
                        ;programului

int 21H                 ;apelul DOS de terminare
                        ;a programului

END ProgramStart        ;directiva de terminare a codului
                        ;programului
```

**Explicații:**

**1.** Se pot introduce comentarii într-un program ASM prin folosirea ";". Tot ce urmează după ";" şi până la sfârşitul liniei este considerat comentariu.

**2.** Nu are importanță dacă programul este scris folosind litere mari sau mici (nu este "case sensitive").

**3.** Fără cele două instrucțiuni care setează registrul DS către segmentul definit de .DATA, funcția de afişare a string-ului nu ar fi funcționat cum trebuie. Variabila DataString se află în segmentul .DATA şi nu poate fi accesată dacă DS nu este poziționat către acest segment. Acest lucru se explică în modul următor: atunci când facem apelul DOS de afişare a unui string, trebuie să parcurgem întreaga adresă de tipul segment:offset a string-ului în DS:DX. De aceea, de abia după ce am încărcat DS cu segmentul .DATA şi DX cu adresa (offset-ul) lui DataString avem o referință completă segment:offset către DataString.

**Observații:**

Nu trebuie să încărcăm în mod explicit registrul de segment CS deoarece DOS face acest lucru automat în momentul când rulăm un program. Astfel, dacă CS nu ar fi deja setat la momentul execuției primei instrucțiuni din program, procesorul nu ar şti unde să găsească instrucțiunea şi programul nu ar rula niciodată. În mod asemănător, registrul de segment SS este setat de DOS înainte de execuția programului şi de regulă rămâne nemodificat pe perioada execuției programului.

Cu registrul de segment DS lucrurile stau altfel. În timp ce registrul CS se referă la intrucțiuni (cod), SS se referă ("pointează") la stivă, DS "pointează" la date. Programele nu manipulează direct instrucțiuni sau stive dar au de-a face în mod direct cu date. De asemenea, programele vor acces la date situate în segmente diferite în orice moment. Se poate dori încărcarea în DS a unui segment, accesarea datelor din acel segment şi apoi încărcarea lui DS cu un alt segment pentru a accesa un bloc diferit de date. În programe mici sau medii nu vom avea nevoie de mai mult de un segment de date dar programe mai complexe folosesc deseori segmente de date multiple.

Următorul program va afişa un caracter pe ecran, folosind încărcarea registrului ES în locul lui DS.

```
;Program p02.asm
.MODEL small
.STACK 200H
.DATA
OutputChar DB 'B'          ;definirea variabilei OutputChar
                           ;inițializată cu valoarea "B"
.CODE
ProgramStart:
mov dx, @data
mov es, dx                 ;spre deosebire de programul anterior,
                           ;se foloseşte ES pentru specificarea
                           ;segmentului de date
mov bx, offset OutputChar  ;se încarcă BX cu adresa
                           ;variabilei OutputChar
mov dl, es:[bx]            ;se încarcă AL cu valoarea de la
                           ;adresa explicită es:[bx]
                           ;(adresare indexată)
mov ah,02                  ;codul funcției DOS de afişare a unui
                           caracter
int 21H                    ;apelul DOS de execuție a afişării
```

```
mov ah, 4cH                ;codul funcției DOS de terminare a
                           ;programului
int 21H                    ;apelul DOS de terminare
                           ;a programului
END ProgramStart           ;directiva de terminare a codului
                           ;programului
```

**DOSSEG** este directiva ce face ca segmentele dintr-un program să fie grupate conform convențiilor Microsoft de adresare a segmentelor.

Directiva **.MODEL**

Este directiva ce specifică modelul de memorie pentru un program ASM ce foloseşte directive de segment simplificate.

Definiții: ***"near"*** înseamnă adresa (offset-ul) pe 16 biți din cadrul aceluiaşi segment, în timp ce ***"far"*** înseamnă o adresă completă de tip ***segment:offset***, din cadrul altui segment decât cel curent.

Modelele de memorie ce se pot specifica prin intermediul directivei .MODEL sunt:

- ***tiny*** - atât codul cât şi datele programului încap în acelaşi segment de 64 Ko. Atât codul cât şi datele sunt de tip ***near***.

- ***small*** - codul programului trebuie să fie într-un singur segment de 64 Ko şi datele într-un bloc separat de 64Ko; codul şi datele sunt ***near***

- ***medium*** - codul programului poate fi mai mare decât 64 Ko dar datele trebuie să fie într-un singur segment de 64 Ko. Codul este ***far***, datele sunt ***near***.

- ***compact*** - codul programului poate fi într-un singur segment, datele pot fi mai mari de 64 Ko. Codul este near, datele sunt ***far***.

- ***large*** - atât codul cât şi datele pot depăşi 64Ko, dar nici un masiv de date nu poate depăşi 64 Ko. Atât codul cât şi datele sunt ***far***.

- ***huge*** - atât codul cât şi datele pot depăşi 64 Ko şi masivele de date pot depăşi 64 Ko. Atât codul cât şi datele sunt ***far***. Pointerii la elementele dintr-un masiv sunt ***far***.

În continuare sunt prezentate câteva exemple legate de modalităţile de declarare a variabilelor şi de adresare a memoriei:

```
var1 DW 01234h          ;se defineşte o variabilă word cu
                        ;valoarea 1234h
var2 DW 01234           ;se defineşte o variabilă word cu
                        ;valoarea zecimală 1234 (4D2 in hexa)
var3 RESW 1             ;se rezervă spaţiu pentru o variabilă
                        ;word (de valoare 0)
var4 DW ABCDh           ;atribuire ilegală!

mesajsco2  DB  'SCO 2 este cursul preferat!'

...start:
  mov ax,cs             ; setarea  segmentului de date
  mov ds,ax             ; DS=CS
; orice referinţă de memorie se presupune că este relativă la segmentul DS
mov ax,[var2]           ; AX <- var2
                        ; == mov ax,[2]

mov si,var2             ;se foloseşte SI ca pointer către var2
                        ;(cod C echivalent SI=&var2)

mov ax,[si]             ;se citeşte din memorie valoarea lui
                        ;var2 (*(&myvar2))
                        ;(referinţă indirectă)

mov bx,mesajsco2        ; BX este pointer la un string
                        ; (cod C echivalent:  BX=&mesajsco2)
```

```
dec BYTE [bx+1]          ; transformă 'C' in 'B' !

mov si, 1                ; Foloseşte SI cu rol de index
inc byte [mesajsco2+SI]  ; == inc byte [SI + 8]
                         ; == inc byte [9]

; Memoria poate fi adresata folosindu-se patru regiştri:
; SI  -> Implică DS
; DI  -> Implică DS
; BX  -> Implică DS
; BP  -> Implică SS !  (nu este foarte des utilizat)
;
```

**Exemple:**

```
mov ax,[bx]              ; ax <- word în memorie referit de BX
mov al,[bx]              ; al <- byte în memorie referit de BX
mov ax,[si]              ; ax <- word referit de SI
mov ah,[si]              ; ah <- byte referit de SI
mov cx,[di]              ; di <- word referit de DI

mov ax,[bp]              ; AX <- [SS:BP] Operație cu stiva!

; În plus, sunt permise BX+SI şi BX+DI:

mov ax,[bx+si]
mov ch,[bx+di]
```

```
; Deplasamente pe 8 sau 16 biti:

mov ax,[23h]          ; ax <- word în memorie DS:0023
mov ah,[bx+5]         ; ah <- byte în memorie [DS:BX+5]
mov ax,[bx+si+107]    ; ax <- word la adresa [DS:BX+SI+107]
mov ax,[bx+di+47]     ; ax <- word la adresa [DS:BX+DI+47]

; ATENȚIE: copierea din memorie în memorie este ilegală!
;Totdeauna trebuie să se treacă valoarea copiată printr-un registru

mov [bx],[si]   ;Ilegal
mov [di],[si]   ;Ilegal

; Caz special: operațiile cu stiva!

pop word [var]        ; var <- [SS:SP]
```

### *3.2.4 Adrese de memorie şi valori*

Un program scris în limbaj de asamblare se poate referi fie la o adresă de memorie (OFFSET = DEPLASAMENT), fie la o valoare stocată de variabilă în memorie. Din păcate, limbajul de asamblare nu este nici strict, nici intuitiv cu privire la modurile în care aceste două tipuri de referire sunt făcute şi, drept urmare, referirile la OFFSET sau la valoare sunt deseori confundate. În figura 3.10 sunt ilustrate conceptele de adresă de deplasament (offset) şi valoare stocată în memorie.

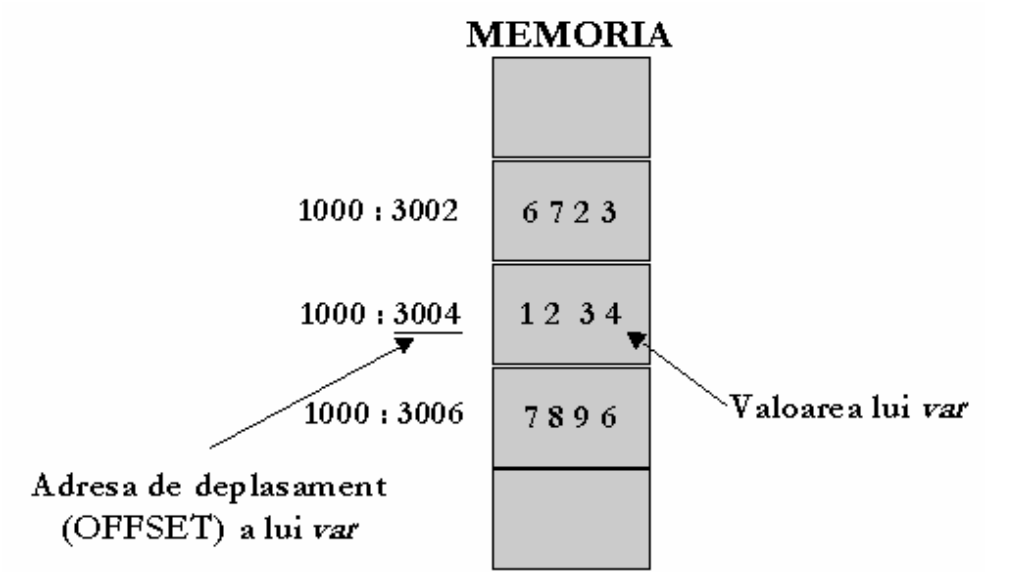



Fig. 3.10 Ilustrarea noțiunilor de adresă de deplasament şi valoare stocată în memorie

Deplasamentul unei variabile de memorie *var* de dimensiune word este valoarea constantă 5004H, obținută cu operatorul OFFSET. Spre exemplu, instrucțiunea:

**MOV BX, OFFSET var**

încarcă valoarea 5004H în registrul BX. Valoarea 5004H nu se modifică; ea este construită în cadrul instrucțiunii. Valoarea lui *var* este 1234H, citită din memorie la adresa dată de offset-ul 5004H din segmentul de date. O modalitatea de citire a acestei valori este de a încărca registrele BX, SI, DI sau BP cu offset-ul lui *var* şi apoi folosirea registrului respectiv pentru adresarea memoriei.

Instrucțiunile:

MOV BX, OFFSET var

MOV AX, [ BX ]

au ca efect încărcarea valorii lui *var* (1234H) în registrul AX.

De asemenea, se poate încărca valoarea lui *var* direct în AX folosind:

MOV AX, var

Sau

MOV AX, [ var ]

În timp ce valoarea deplasamentului rămâne constantă, valoarea 1234H nu este permanent asociată cu *var*. De exemplu, instrucțiunile:

MOV [ var ], 5555H

MOV AX, [ var ]

au ca efect încărcarea valorii 5555H în registrul AX.

Cu alte cuvinte, în timp ce deplasamentul lui *var* este o valoare constantă ce descrie o adresă fixă dintr-un segment de date, valoarea variabilei *var* este un număr ce poate fi modificat şi care se află memorat la adresa (de memorie) respectivă.

Instrucțiunile:

MOV [ var ], 1

ADD [ var ], 2

modifică valoarea lui var la 3, dar instrucțiunea:

**ADD OFFSET var, 2** este echivalentă cu **ADD 5002H, 2**, ceea ce este un lucru fără sens, deoarece este imposibil să se însumeze o constantă cu alta.

O problemă ce poate apărea adesea în timpul programării este aceea a omiterii lui OFFSET; de exemplu, dacă scriem **MOV SI, var** atunci când, de fapt, dorim încărcarea în SI a deplasamentului lui *var*. Nu va fi semnalată nici o eroare în acest caz, având în vedere că *var* este o variabilă de tip word. Totuşi, în momentul execuției programului, registrul SI va fi încărcat cu valoarea lui *var* (1234H), în loc de OFFSET, ceea ce poate conduce la rezultate imprevizibile. În acest caz, referirile la constantele de adresă vor fi precedate de OFFSET iar referirile la valori din memorie să fie cuprinse între paranteze drepte („[” şi „]”), eliminând astfel ambiguitatea.

### *3.2.5 Instrucțiuni ale microprocesorului Intel*

Microprocesoarele din familia Intel x86 dispun de o serie impresionantă de instrucțiuni, asemeni tuturor procesoarelor din clasa procesoarelor CISC (Complex Instruction Set Computer). Instrucțiunile se pot împărți în: instrucțiuni logice, aritmetice, de transfer şi de control. Prezentăm în continuare câteva exemple din fiecare clasă de instrucțiuni.

#### *3.2.5.1 Instrucțiuni logice*

Instrucțiunile logice implementează funcțiile logice de bază, pe un octet sau pe cuvânt. Ele acționează bit cu bit, deci se aplică funcția logică respectivă tuturor biților sau perechilor de biți corespunzători operanzilor. Instrucțiunile logice sunt următoarele:

- **NOT**: A =~A
- **AND**: A &= B
- **OR**: A |= B
- **XOR**: A ^= B
- **TEST**: A & B

De regulă, instrucțiunile logice au efect asupra indicatorilor de stare, cu excepția instrucțiunii NOT, care nu are efect asupra nici unui flag (indicator de stare). Aceste efecte sunt următoarele:

- Se şterge indicatorul carry (C)
- Se şterge indicatorul overflow (O)
- Se setează zero flag (Z) dacă rezultatul este zero, sau îl şterge în caz contrar
- Se copiază bitul mai "înalt" al rezultatului în indicatorul sign (S)
- Se setează bitul de paritate (P) conform cu *paritatea* rezultatului

**Instrucțiunea NOT**

Este o instrucțiune cu un singur operand (instrucțiune *unară*), cu forma generală:

NOT *destinatie*

Unde *destinație* este fie un registru, fie o locație de memorie pe 8 sau 16 biți. Instrucțiunea are ca efect inversarea (negarea) tuturor biților operandului, adică aducerea în forma codului invers - complement față de 1.

**Instrucțiunea AND**

Este o instrucțiune cu doi operanzi (instrucțiune *binară*), cu forma generală:

**AND *destinație, sursa***

Unde *destinatie* este fie un registru, fie o locație de memorie pe 8 sau 16 biți, iar *sursa* poate fi registru, locație de memorie sau o constantă pe 8 sau 16 biți. Instrucțiunea are ca efect operația: <destinație> == <destinație> AND <sursă>. Indicatorii de stare modificați sunt: SF, ZF, PF, CF, OF = 0, AF nedefinit.

Instrucțiunea TEST (AND "non-distructiv")

Este o instrucțiune cu doi operanzi (instrucțiune *binară*), cu forma generală:

**TEST *destinație, sursa***

Unde *destinatie* este fie un registru, fie o locație de memorie pe 8 sau 16 biți, iar *sursa* poate fi registru, locație de memorie sau o constantă pe 8 sau 16 biți. Instrucțiunea are acelaşi efect ca şi instrucțiunea AND, cu deosebirea că nu se modifică operandul destinație, iar indicatorii de stare sunt modificați în acelaşi mod ca şi în cazul instrucțiunii AND.

**Instrucțiunea OR**

Este o instrucțiune cu doi operanzi, cu forma generală:

**OR *destinație, sursa***

Unde *destinație* este fie un registru, fie o locație de memorie pe 8 sau 16 biți, iar *sursa* poate fi registru, locație de memorie sau o constantă pe 8 sau 16 biți. Instrucțiunea are efectul: <destinație> == <destinație> OR <sursă>. Indicatorii de stare modificați sunt: SF, ZF, PF, CF, OF = 0, AF nedefinit.

**Instrucțiunea XOR (SAU-Exclusiv)**

Este o instrucțiune cu doi operanzi, cu forma generală:

**XOR *destinație, sursa***

Unde *destinație* este fie un registru, fie o locație de memorie pe 8 sau 16 biți, iar *sursa* poate fi registru, locație de memorie sau o constantă pe 8 sau 16 biți. Instrucțiunea are efectul: <destinație> == <destinație> XOR <sursă>. Indicatorii de stare modificați sunt: SF, ZF, PF, CF, OF = 0, AF nedefinit. Funcția XOR, denumită SAU-Exclusiv (sau *anticoincidență*) are valoarea logică 1 atunci când operanzii săi sunt diferiți (unul are valoarea 0 iar celălalt valoarea 1) şi valoarea logică 0 când ambii operanzi au aceeaşi valoare (fie ambii au valoarea 0, fie ambii au valoarea 1).

**Observație:**

De cele mai multe ori, instrucțiunile AND şi OR sunt folosite pe post de „mascare" a datelor; în acest sens, o valoarea de tip „mască" (*mask*) este utilizată pentru a forța anumiți biți să ia valoarea zero sau valoarea 1 în cadrul altei valori. O astfel de „mască" logică are efect asupra anumitor biți, în timp ce pe alții îi lasă neschimbați. Exemple:

- Instrucțiunea **AND CL, 0Fh** – face ca cei mai semnificativi 4 biți să ia valoarea 0, în timp ce biții mai puțin semnificativi sunt lăsați neschimbați; astfel, dacă registrul CL are valoarea inițială **1001 1101**, după execuția instrucțiunii **AND CL, 0Fh** va avea valoarea **0000 1101**.

- Instrucțiunea **OR CL, 0Fh** – face ca cei mai puțin semnificativi 4 biți să ia valoarea 1, în timp ce biții mai semnificativi să rămână nemodificați. Dacă registrul CL are valoarea inițială **1001 1101**, după execuția instrucțiunii **OR CL, 0Fh** va avea valoarea **1001 1111**.

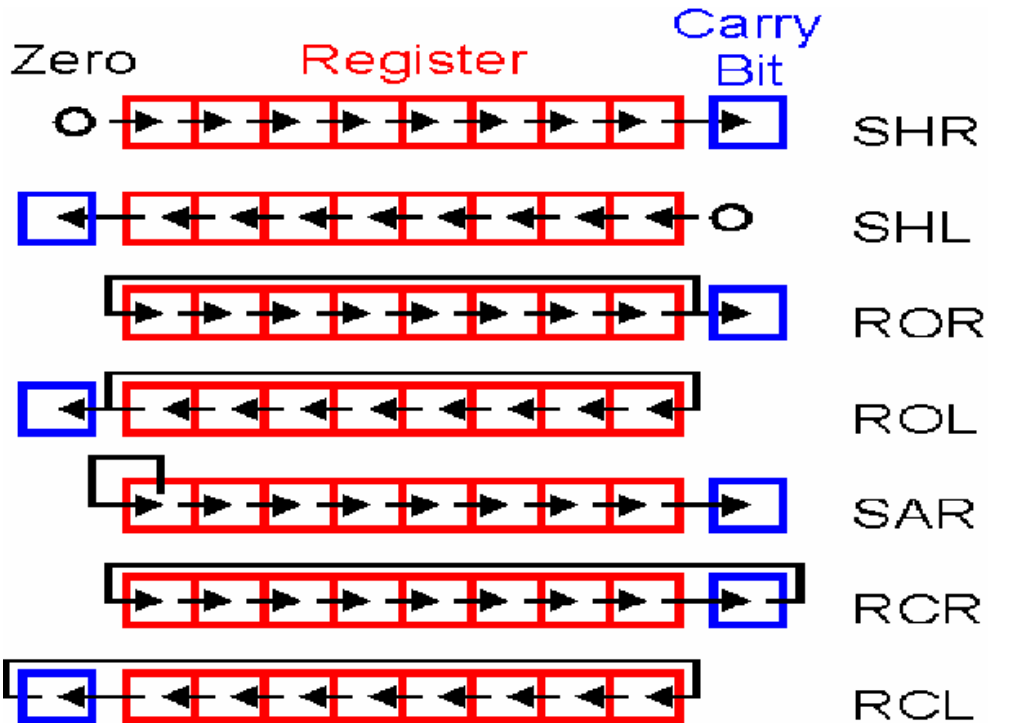



Fig. 3.11 Instrucțiuni de deplasare şi de rotație

### *3.2.5.2 Instrucțiuni de deplasare şi de rotație*

Acest tip de instrucțiuni (vezi figura 3.11) permit realizarea operațiilor de deplasare şi de rotație la nivel de bit. Ele au doi operanzi, primul operand fiind cel asupra căruia se aplică operația de deplasare pe biți, iar cele de-al doilea (operandul *numărător* sau *contor*) semnifică numărul de biți cu care se face această deplasare. Operațiile se pot face de la dreapta spre stânga sau invers. Deplasarea înseamnă translatarea tuturor biților din operand la stânga/dreapta, cu completarea unei valori fixe în poziția rămasă liberă şi cu pierderea biților din dreapta/stânga. Rotația presupune translatarea biților din operand la stânga/dreapta, cu completarea în dreapta/stânga cu biții care se pierd în partea opusă. Sintaxa generală a instrucțiunilor de deplasare şi rotație este următoarea:

**INSTR <operand> , <contor>**

Unde **INSTR** reprezintă numele instrucțiunii, **<operand>** reprezintă un registru sau o locație de memorie pe 8 sau 16 biți, iar **<contor>** semnifică numărul de biți cu care se face deplasarea, adică fie o constantă, fie registrul CL (care îşi confirmă astfel rolul de numărător).

**Observație**

Totdeauna există două modalități de deplasare:

- Prin folosirea unui contor efectiv – de exemplu: SHL AX, 1
- Prin folosirea registrului CL pe post de contor – de exemplu: SHL AX, CL

**Instrucțiunea SHL/SAL (Shift Left/Shift Arithmetic Left)**

Această instrucțiune translatează biții operandului o poziție la stânga de câte ori specifică operandul numărător. Pozițiile rămase libere prin deplasarea la stânga sunt umplute cu zerouri la bitul cel mai puțin semnificativ, în timp ce bitul cel mai semnificativ se deplasează în indicatorul CF (Carry Flag).

Reprezintă o modalitate rapidă de înmulțire cu o putere a lui 2 (în funcție de numărul de biți pentru care se face deplasarea la stânga).

**Exemple:**

**1.** Înmulțirea lui AX cu 10 (1010 în binar) (înmulțim cu 2 şi cu 8, apoi adunăm rezultatele)

```
SHL   ax, 1        ; AX ori 2
MOV   bx, ax       ; salvăm 2*AX în BX
SHL   ax, 2        ; 2*AX(original) * 4 = 8*AX(original)
ADD   ax, bx       ; 2*AX + 8*AX = 10*AX
```

**2.** Înmulțirea lui AX cu 18 (10010 în binar) (înmulțim cu 2 şi cu 16, apoi adunăm rezultatele)

```
SHL   ax, 1        ; AX ori 2
MOV   bx, ax       ; salvăm 2*AX
SHL   ax, 3        ; 2*AX(original) ori 8 = 16*AX(original)
ADD   ax, bx       ; 2*AX + 16*AX = 18*AX
```

**Instrucțiunea SHR (Shift Right)**

Această instrucțiune translatează biții operandului o poziție la dreapta de câte ori specifică operandul numărător. Bitul cel mai puțin semnificativ se deplasează în indicatorul CF (Carry Flag).

Reprezintă o modalitate rapidă de împărțire *fără semn* la o putere a lui 2 (dacă deplasarea se face cu *o poziție* la dreapta, operația este echivalentă cu o împărțire la 2, dacă deplasarea se face cu două poziții, operația este echivalentă cu o împărțire la $2^2$, etc.). Operația de împărțire se execută *fără semn*, completându-se cu un bit 0 dinspre stânga (bitul cel mai semnificativ).

**Instrucțiunea SAR (Shift Arithmetic Right)**

Această instrucțiune translatează biții operandului o poziție la dreapta de câte ori specifică operandul numărător. Bitul cel mai semnificativ rămâne neschimbat, în timp ce bitul cel mai puțin semnificativ este copiat în indicatorul CF (Carry Flag).

Reprezintă o modalitate rapidă de împărțire *cu semn* la o puterea a lui 2 (în funcție de numărul de biți cu care se face deplasarea la dreapta).

**Instrucțiunea RCL (Rotate through Carry Left)**

Această instrucțiune determină o rotație a biților operandului către stânga prin intermediul lui CF (Carry Flag). Astfel, cel mai semnificativ bit trece din operand în CF, apoi se deplasează toți biții din operand cu o poziție la stânga iar CF original trece în bitul cel mai puțin semnificativ din operand.

**Instrucțiunea ROL (Rotate Left)**

Această instrucțiune determină o rotație a biților operandului către stânga. Astfel, cel mai semnificativ bit trece din operand în bitul cel mai puțin semnificativ.

**Exemplu.**

După execuția instrucțiunilor:

ROL AX, 6

AND AX, 1Fh

Biții 10-14 din AX se mută în biții 0-4.

**Instrucțiunea RCR (Rotate through Carry Right)**

Această instrucțiune determină o rotație a biților operandului către dreapta prin intermediul lui CF (Carry Flag). Astfel, bitul din CF este scris înapoi în bitul cel mai semnificativ al operandului.

**Instrucțiunea ROR (Rotate Right)**

Această instrucțiune determină o rotație a biților operandului către dreapta. Bitul cel mai puțin semnificativ trece în bitul cel mai semnificativ.

**Exemple:**

```
MOV ax,3        ; Valori inițiale                AX = 0000 0000 0000 0011
MOV bx,5        ;                                BX = 0000 0000 0000 0101
OR ax,9         ; ax <- ax | 0000 1001           AX = 0000 0000 0000 1011
AND ax,10101010b  ; ax <-ax&1010 1010 AX = 0000 0000 0000 1010
XOR ax,0FFh ; ax <- ax ^ 1111 1111               AX = 0000 0000 1111 0101
NEG ax          ; ax <- (-ax)                    AX = 1111 1111 0000 1011
NOT ax          ; ax <- (~ax)                    AX = 0000 0000 1111 0100
OR ax,1         ; ax <- ax | 0000 0001           AX = 0000 0000 1111 0101
SHL ax,1        ; depl logică la stg cu 1 bit    AX = 0000 0001 1110 1010
SHR ax,1        ; depl logică la dr cu 1 bit     AX = 0000 0000 1111 0101
ROR ax,1        ; rotație stg (LSB=MSB)          AX = 1000 0000 0111 1010
ROL ax,1        ; rotație dr (MSB=LSB)           AX = 0000 0000 1111 0101
MOV cl,3        ; folosim CL pt depl cu 3 biți   CL = 0000 0011
SHR ax,cl       ; împărțim AX la 8               AX = 0000 0000 0001 1110
MOV cl,3        ; folosim CL pt depl cu 3 biți   CL = 0000 0011
SHL bx,cl       ; înmulțim BX cu 8               BX = 0000 0000 0010 1000
```

### *3.2.5.3 Instrucțiuni aritmetice*

**Instrucțiunea ADD (ADDition)**

Instrucțiunea ADD are formatul general:

**ADD <destinație> <sursă>**

Unde <*destinație*> poate fi un registru general sau o locație de memorie, iar <*sursă*> poate fi registru general, locație de memorie sau o valoare imediată. Cei doi operanzi nu pot fi însă în acelaşi timp locații de memorie. Rezultatul operației este următorul: <destinație> == <destinație> + <sursă>. Indicatorii de stare modificați în urma acestei operații sunt: AF, CF, PF, SF, ZF, OF. Operanzii pot fi pe 8 sau pe 16 biți şi trebuie să aibă aceeaşi dimensiune. Dacă apare ambiguitate la modul de exprimare al operanzilor (8 sau 16 biți) se va folosi operatorul PTR.

**Exemple:**

```
ADD  AX, BX          ; adunare între regiştri – AX ← AX + BX
ADD  DL, 33h         ; adunare efectivă - DL ← DL + 33h
MOV  DI, NUMB        ; adresa lui NUMB
MOV  AL, 0           ; se şterge suma
ADD  AL, [DI]        ; adună [NUMB]
ADD AL, [DI + 1]     ; adună [NUMB + 1]
ADD word ptr [DI], -2        ; destinație în memorie, sursă imediată
ADD byte ptr VAR, 5          ; forțarea instrucțiunii pe un octet,
                             ;VAR fiind  declarat DW
```

**Instrucțiunea INC (Increment addition)**

Instrucțiunea INC are formatul general:

**INC <destinație>**

Unde <*destinație*> este un registru sau un operand în memorie, pe 8 sau pe 16 biți iar semnificația operației este incrementarea valorii destinație cu 1. Toți indicatorii de stare sunt afectați, cu excepția lui CF (Carry Flag).

**Exemplu:**

```
MOV   DI, NUMB          ; adresa lui NUMB
MOV   AL, 0             ; şterge suma
ADD   AL, [DI]          ; adună [NUMB]
INC   DI                ; DI = DI + 1
ADD AL, [DI]            ; adună [NUMB + 1]
```

**Instrucțiunea ADC (ADdition with Carry)**

Instrucțiunea ADD are formatul general:

**ADD <destinație> <sursă>**

Unde <*destinație*> poate fi un registru general sau o locație de memorie, iar <*sursă*> poate fi registru general, locație de memorie sau o valoare imediată. Instrucțiunea acționează întocmai ca ADD, cu deosebirea că la rezultat este adăugat şi bitul CF. Este utilizat, de regulă, pentru a aduna numere mai mari de 16 biți (8086-80286) sau mai mari de 32 de biți la 80386, 80486, Pentium.

**Exemplu:**

Adunarea a două numere pe 32 de biți se poate face astfel (BXAX) + (DXCX):

```
ADD   AX, CX
ADC   BX, DX
```

**Instrucțiunea SUB (SUBstract)**

Instrucțiunea SUB are formatul general:

**SUB <destinație> <sursă>**

Unde <*destinație*> poate fi un registru general sau o locație de memorie, iar <*sursă*> poate fi registru general, locație de memorie sau o valoare imediată. Rezultatul operației este următorul: <destinație> == <destinație> - <sursă>. Indicatorii de stare modificați în urma acestei operații sunt: AF, CF, PF, SF, ZF, OF. Operanzii pot fi pe 8 sau pe 16 biți şi trebuie să aibă aceeaşi dimensiune. Scăderea poate fi văzută ca o adunare cu reprezentarea în complementul față de 2 al operandului sursă şi cu

inversarea bitului CF, în sensul că, dacă la operație (adunarea echivalentă) apare transport, CF=0 şi dacă la adunarea echivalentă nu apare transport, CF=1.

Pentru instrucțiunile:

MOV CH, 22h

SUB CH, 34h

Rezultatul este **-12 (1110 1110)**, iar indicatorii de stare se modifică astfel:

ZF = 0 (rezultat diferit de zero)

CF = 1 (împrumut)

SF = 1 (rezultat negativ)

PF = 0 (paritate pară)

OF = 0 (fără depăşire)

**Instrucțiunea DEC (DECrement substraction)**

Instrucțiunea DEC are formatul general:

**DEC <destinație>**

Unde <*destinație*> este un registru sau un operand în memorie, pe 8 sau pe 16 biți iar semnificația operației este decrementarea valorii destinație cu 1. Toți indicatorii de stare sunt afectați, cu excepția lui CF (Carry Flag).

**Instrucțiunea SBB (SuBstract with Borrow)**

Instrucțiunea SBB are formatul general:

**SBB <destinație>, <sursă>**

Unde <*destinație*> şi <*sursă*> pot fi registru sau operand în memorie, pe 8 sau pe 16 biți. Rezultatul operației este următorul: <destinație> == <destinație> - <sursă> - CF, deci la fel ca şi în cazul instrucțiunii SUB, dar din rezultat se scade şi bitul CF. Indicatorii de stare modificați în urma acestei operații sunt: AF, CF, PF, SF, ZF, OF. Această instrucțiune este utilizată, de regulă, pentru a scădea numere mai mari de 16 biți (la 8086 - 80286) sau de 32 de biți (la 80386, 80486, Pentium).

**Exemplu**

Scăderea a două numere pe 32 de biți se poate face astfel (BXAX) - (SIDI):

```
SUB    AX, DI
SBB    BX, SI
```

**Setarea indicatorilor de stare în cazul situațiilor de depăşire sau/şi transport**

Indicatorul CF (Carry Flag) indică apariția unui transport în urma unei adunări sau un împrumut în cazul unei scăderi. În cazul unei operații fără semn, CF este setat (ia valoarea 1) atunci când apare transport şi nu este setat (are valoarea 0) dacă nu apare transport. De exemplu, dacă adunăm valorile 36864 (9000h) + 36864 (9000h) = 73728 (12000h) > 65535 (FFFFh), rezultatul depăşeşte valoarea maximă 65535, deci CF va avea valoarea 1. În concluzie, bitul carry este setat atunci când numărul fără semn iese din domeniu (înseamnă că operația aritmetică a generat depăşire).

De asemenea, poate să apară situația de depăşire (indicatorul OF – Overflow Flag este setat) atunci când se adună sau se scad numere fără semn. De exemplu, pentru adunarea 20480 (5000h) + 20480 (5000h) = 40960 (A000h) > 32767 (7FFFh) indicatorul OF este setat, iar CF nu este setat (avem depăşire, fără transport). Indicatorul OF este setat atunci când numărul cu semn depăşeşte domeniul de definiție. De exemplu, în cazul FFFFh + FFFFh = FFFEh = (-1) + (-1) = -2, rezultatul operației este fără depăşire, cu transport. În figura 3.12 putem vedea cum se realizează setarea indicatorilor de stare.

**Alte exemple:**

```
MOV   ax, 0fffeh        ; 65534 interpretat ca fără semn
ADD   ax, 3             ; C = 1, O = 0  ---  Condiție de overflow
                  ; fără semn

MOV   ax, 0FFFEh        ; -2 interpretat ca număr cu semn
ADD   ax, 3             ; C = 1, O = 0  ---  Ok - nr cu semn
MOV   bx, 07FFFh        ; 32767 interpretat ca nr cu semn
ADD   bx, 1             ; C = 0, O = 1  ---  Condiție de Overflow
                  ; cu semn

MOV   bx, 07FFFh        ; 32767 interpretat ca fără semn
ADD   bx, 1             ; C = 0, O = 1  ---  Ok - fără semn
MOV   ax, 07h  ; 7 interpretat fie cu semn fie fără semn
ADD   ax, 03h  ; C = 0, O = 0  ---  Ok indiferent de semn
```

### *3.2.5.4 Instrucțiuni de salt*

Instrucțiunea de salt necondiționat JMP

Această instrucțiune are sintaxa generală:

JMP etichetaUnde *eticheta* indică adresa (locul) unde se face saltul, de regulă prin specificarea unei etichete ce indică un nume de procedură, o etichetă definită cu : sau o etichetă definită cu directiva LABEL. Exemple de etichete:

```
eticheta1:
contor:
eticheta2 LABEL FAR
stop LABEL NEAR
```

Din punct de vedere al saltului există trei tipuri de instrucțiuni JMP:

- *Short jump* - instrucțiune pe 2 octeți ce permite saltul la locația de memorie în intervalul +127 şi -128 octeți de la locația de memorie ce urmează saltului – are forma: **JMP SHORT *eticheta***
- *Near jump* - instrucțiune pe 3 octeți ce permite saltul la locația de memorie în intervalul +/-32KB de la locația de instrucțiunii din segmentul de cod curent – are forma: **JMP *eticheta***
- *Far jump* - instrucțiune pe 5 octeți ce permite saltul la o locație de memorie în întreg spațiul de memorie – are forma: **JMP *eticheta***Pentru microprocesoarele 80386 şi 80486 saltul de tip *near* este în spațiul de memorie +/-2G dacă computerul operează în mod protejat şi în domeniul +/-32K bytes pentru modul real.

**Instrucțiuni de salt condițional**

După cum am văzut mai înainte, instrucțiunile aritmetice şi logice determină modificarea indicatorilor de stare; cu alte cuvinte, aceşti indicatori ne oferă permanent informații cu privire la rezultatele instrucțiunilor anterioare. Prin utilizarea valorilor indicatorilor de stare se pot realiza aşa numitele salturi condiționale, în care se realizează transferul execuției către o altă instrucțiune din cadrul programului. Dacă o condiție de testare a fost adevărată atunci se face un salt înainte sau înapoi către locația specificată (pointerul de instrucțiune IP ne indică în fiecare moment adresa următoarei instrucțiuni ce va fi executate). Dacă acea condiție de testare a fost falsă, se continuă execuția programului cu următoarea instrucțiune (IP este incrementat ca de obicei).

Salturile condiționale sunt întotdeauna de tip *short* pentru 8086-80286; domeniul saltului este între 127 octeți şi -128 octeți de la locația următoare saltului condițional. De la 80386, salturile condiționale sunt fie *short*, fie *near*. Instrucțiunile de salt condițional testează următorii indicatori de stare: SF (Sign Flag), ZF(Zero Flag), CF(Carry Flag), PF (Parity Flag) şi OF (Overflow Flag).

| FLAG | Nume | Descriere | Observaţii |
|---|---|---|---|
| ZF | Zero | 1:ZR:Zero<br>0:NZ: Non-zero | 1 indică faptul că rezultatul este zero<br>0 altfel |
| CF | Carry | 1:CY<br>0:NC | **Operaţii matematice fără semn** şi deplasări (transport sau împrumut) |
| OF | Overflow | 1:OV<br>0:NV | **Operaţii matematice cu semn**<br>Nr pozitive sau negative reprezentate în complement faţă de 2 |
| SF | Sign Flag | 1:NG: -<br>0:PL: + | Bitul cel mai semnificativ al rezultatului |

Fig. 3.12 Setarea indicatorilor de stare

**Observaţie:** Atenţie la compararea valorii FFh cu valoarea 00h! În mulţimea numerelor fără semn prima valoare este mai mare, pe când în mulţimea numerelor cu semn prima valoare este mai mică (are valoarea -1).

Înainte de a prezenta instrucţiunile de salt condiţional vom prezenta însă o instrucţiune care este folosită deseori în compania unei instrucţiuni de salt condiţional: instrucţiunea CMP.

**Instrucţiunea CMP (CoMParison)**

Această instrucţiune compară două valori X şi Y, fiind echivalentă cu o scădere care modifică doar indicatorii de stare. Sintaxa generală este:

**CMP *destinaţie sursă***

Semnificaţia este execuţia unei scăderi temporare *destinaţie – sursă*, cu modificarea indicatorilor de stare. În cazul acestei instrucţiuni toţi indicatorii de stare sunt afectaţi, fiind folosită pentru verificarea conţinutului unui registru sau a unei locaţii de memorie cu o altă valoare. De regulă, această instrucţiune se foloseşte împreună cu o instrucţiune de salt condiţional. În figura 3.13 sunt prezentate modificările indicatorilor de stare rezultate în urma instrucţiunii CMP.

**Exemplu:**

CMP al, 10h ; compară conţinutul registrului AL cu 10h

JAE SUPER ; dacă AL are valoarea 10h sau mai mult se face salt la
; locatia de memorie SUPER

CMP Op1, Op2

| Operanzi fără semn | Operanzi cu semn |
|---|---|
| **Z**: egalitate/inegalitate | **Z**: egalitate/inegalitate |
| **C**: Op1 < Op2 (C=1)<br>Op1 >= Op2 (C=0) | **C**: fără semnificaţie |
| **S**: fără semnificaţie<br>**O**: fără semnificaţie | **S** şi **O** împreună<br>Dacă ((S=0) şi (O=1)) sau ((S=1) şi (O=0))<br>atunci Op1 < Op2<br>Dacă ((S=0) şi (O=0)) sau ((S=1) şi (O=1))<br>atunci Op1 >= Op2 |

Fig. 3.13 Setarea indicatorilor de stare în urma execuției instrucțiunii CMP

Compararea întregilor cu semn cu ajutorul instrucțiunii CMP se face astfel (considerăm instrucțiunea CMP AX, BX):

- Indicatorul SF va fi setat în funcție de rezultatul AX-BX (dacă are valoarea 1 în bitul cel mai semnificativ al rezultatului).
- Indicatorul OF va fi setat dacă rezultatul AX-BX a produs un număr în afara domeniului de definiție (de exemplu -32768 ÷ +32767 pentru o reprezentare pe16 biți) reprezentat ca întreg.

Pentru instrucțiunile de salt condițional există mai multe mnemonice pentru aceeaşi instrucțiune şi dacă condiția nu este îndeplinită, saltul nu are loc şi se continuă cu execuția instrucțiunii următoare din program. În figurile 3.14 şi 3.15 sunt prezentate instrucțiunile de salt condiționat.

| Comandă | Descriere | Condiţie |
|---|---|---|
| JA=JNBE | Jump if above | C=0 & Z=0 |
| | Jump if not below or equal | |
| JBE=JNA | Jump if below or equal | C=1 \| Z=1 |
| JAE=JNB=JNC | Jump if above or equal | C=0 |
| | Jump if not below | |
| | Jump if no carry | |
| JB=JNA=JC | Jump if below | C=1 |
| | Jump if carry | |
| JE=JZ | Jump if equal | Z=1 |
| | Jump if Zero | |
| JNE=JNZ | Jump if not equal | Z=0 |
| | Jump if not zero | |
| JS | Jump Sign (MSB=1) | S=1 |

Fig. 3.14 Instrucțiuni de salt condiționat

O altă instrucțiune de salt este instrucțiunea LOOP, cu formatul general:

**LOOP *eticheta***

unde *eticheta* este o locație de memorie în program. Instrucțiunea LOOP reprezintă o combinație de decrementare a registrului general CX cu un salt condițional; mai precis, în prima fază LOOP decrementează CX (sau ECX pentru modul de lucru pe 32 de biți), iar dacă CX este diferit de 0 se face salt la adresa indicată de *eticheta*. În cazul în care registrul CX este 0, se va executa următoarea instrucțiune din program (se continuă execuția în mod normal, secvențial).

| Comandă | Descriere | Condiţie |
|---|---|---|
| JNS | Jump Not Sign (MSB=0) | S=0 |
| JO | Jump if overflow set | O=1 |
| JNO | Jump if no overflow | O=0 |
| JG=JNLE | Jump if greater | |
| | Jump if not less or equal | S=O & Z=0 |
| JGE=JNL | Jump if greater or equal | S=O |
| | Jump if not less | |
| JL=JNGE | Jump if less | S^O |
| | Jump if not greater or equal | |
| JLE=JNG | Jump if less or equal | S^O \| Z=1 |
| | Jump if not greater | |
| JCXZ | Jump if register CX=zero | CX=0 |

Fig. 3.15 Instrucțiuni de salt condiționat

### 3.3 Exemple de programe

**1**. Program care citeşte un număr de la tastatură şi afişează dacă numărul este par sau nu.

```
; Programul citeşte un număr şi afişează un mesaj referitor la paritate
dosseg
.model small
.stack
.data
mesaj  db 13,10,'Introduceți numărul:(<=9)$'
mesg_par  db 13,10,'Numărul introdus este par!$'
mesg_impar  db 13,10,'Numărul introdus este impar!$'
.code
pstart:
        mov ax,@data
        mov ds,ax
        mov ah,09
        mov dx,offset mesaj
        int 21h
```

```
mov ah,01h ; se citeşte un caracter de la tastatură
           ; codul ASCII al caracterului introdus va fi în AL
           int 21h
           mov bx,2
           div bx ; se împarte AX la BX, câtul va fi în AX, restul în DX
           cmp dx,0
           jnz impar
           mov ah,09
           mov dx,offset mesg_par
           int 21h
           jmp sfârşit
impar:     mov ah,09
           mov dx,offset mesg_impar
           int 21h
sfarsit:
           mov ah,4ch
           int 21h ; sfârşitul programului
END pstart
```

**2**. Program care calculează pătratul unui număr introdus de la tastatură.

```
; Programul calculează pătratul unui număr (<=256) introdus de la tastatură
; Valoarea pătratului se calculează în registrul AX (valoare maximă 2^16 = 65536)
dosseg
.model small
.stack
.data

nr DB 10,10 dup(0)
r DB 10, 10 dup(0)

mesaj  db 13,10,'Introduceţi numărul:(<=256)$'
pătrat db 13,10,'Pătratul numărului este:$'

.code

pstart:
      mov ax,@data
```

```
        mov ds,ax

        mov ah,09
          mov dx,offset mesaj
        int 21h

          mov ah,0ah
          mov dx,offset nr
          int 21h

          mov cl,nr[1]   ; încarc în CL numărul de cifre al numărului introdus
          inc cl         ; în şir se va merge până la poziția cl+1
          mov si,1       ; folosesc registrul SI pe post de contor
          xor ax,ax      ; inițializez AX cu valoarea 0
          mov bl,10      ; se va înmulți cu valoarea 10 care este stocată în BL
    înmultire:
          mul bl
          inc si
          mov dl,nr[si]
          sub dl,30h
          add ax,dx
          cmp si,cx
          jne înmulțire

          mul ax
          xor si,si
          mov bx,10

    cifra: ; aici începe afişarea rezultatului din AX
          div bx
          add dl,30h
          mov r[si],dl
          inc si
          xor dx,dx
          cmp ax,0
          jne cifra

          mov ah,9
          mov dx, offset pătrat
```

```
                int 21h

        caracter:
                dec si
                mov ah,02       ;apelarea funcției 02 pentru afişarea unui
caracter
                mov dl,r[si]    ;al carui cod ASCII este în DL
                int 21h
                cmp si,0
                jne caracter
                jmp sfârşit

                mov ah,9
                mov dx,offset pătrat
                int 21h
        sfârşit:
                mov ah,4ch
                int 21h ; stop program
        END pstart
```

**3**. Program care calculează valoarea unui număr ridicat la o putere. Atât numărul cât şi exponentul (puterea) sunt introduse de la tastatură.

```
        ; Programul calculează un număr ridicat la o putere
        ; Observație. Deoarece rezultatul se calculează în registrul AX care
;este un registru pe 16 biți, valoarea maximă calculată corect este
2^16= ;65536

.model small
.stack
.data
mesaj1  db 13,10,'Introduceți numărul:(<=9)$'
mesaj2  db 13,10,'Introduceți puterea:(<=9)$'
mesaj_final  db 13,10,'Rezultatul este: $'
mesaj_putere_0 db 13,10, 'Orice număr ridicat la puterea 0 este 1! $'
r db 30 dup(0) ; în variabila r se va stoca rezultatul
.code
pstart:
        mov ax,@data
        mov ds,ax
```

```
mov ah,09
mov dx,offset mesaj1
int 21h
mov ah,01h      ; se citeşte un caracter de la tastatură
                ; codul ASCII al caracterului introdus va fi în AL
int 21h
and ax,00FFh
sub ax, 30h     ; se obţine valoarea numerică
                ; scăzându-se codul lui 0 în ASCII (30H)
push ax         ; se salvează valoarea lui ax în stivă
mov ah,09
mov dx,offset mesaj2
int 21h
mov ah,01h      ; se citeşte un caracter de la tastatură
                ; codul ASCII al caracterului introdus va fi în AL
int 21h
and ax,00FFh
sub ax, 30h     ; se obţine valoarea numerică
                ; scăzându-se codul lui 0 în ASCII (30H)
mov cx,ax       ;registrul CX contorizează numărul de înmulţiri
cmp cx,0
jne putere_0
mov ah,09
mov dx, offset mesaj_putere_0
int 21h
jmp sfârşit
putere_0:
pop bx          ;se salvează în BX valoarea cu care înmulţeşte
mov ax,0001
înmulţire:
        mul bx
        loop înmulţire
        xor si,si
        mov bx,10
cifra:
        div bx
        add dl,30h
        mov r[si],dl
        inc si
        xor dx,dx
        cmp ax,0
```

```
        jne cifra
        mov ah,9
        mov dx, offset mesaj_final
        int 21h
caracter:
dec si
mov ah,02       ;apelarea funcţiei 02 pentru afişarea unui caracter
mov dl,r[si]    ;al cărui cod ASCII este în DL
int 21h
cmp si,0
jne caracter
sfârşit:
mov ah,4ch
int 21h         ; sfârşitul programului
END pstart
```

**4**. Program care verifică dacă un număr este palindrom (un număr se numeşte palindrom dacă scris de la dreapta la stânga sau invers are aceeaşi valoare).

```
; Programul verifică dacă un număr sau şir de caractere este
;palindrom

dosseg
.model small
.stack
.data

nr DB 10,10 dup(0)
mesaj  db 13,10,'Introduceţi numărul:$'
mesaj_nu db 13,10,'Numărul nu este palindrom!$'
mesaj_da db 13,10,'Numărul este palindrom!$'
.code
pstart:
      mov ax,@data
      mov ds,ax
      mov ah,09
      mov dx,offset mesaj
      int 21h
         mov ah,0ah
         mov dx,offset nr
```

```
        int 21h
        mov si,1
    mov cl,nr[si]   ; încarc în CL numărul de cifre al numărului introdus
    and cx,00FFh
    mov ax,cx
    mov bl,2
    div bl          ; în AL este catul împărțirii lui AX la 2
    and ax,00FFh
    inc ax
    inc cx
    mov di,cx
    următorul_caracter:
    inc si          ; SI creşte de la începutul şirului spre mijloc
    mov bl,nr[di]
    cmp nr[si],bl
    jne nu_este
    dec di          ; DI scade de la sfârşitul şirului spre mijloc
    cmp si,ax       ; în şir se va merge până la poziția cl+1
    jne următorul_caracter
    mov ah,9
    mov dx,offset mesaj_da
    int 21h
    jmp sfârşit
    nu_este:
    mov ah,9
    mov dx,offset mesaj_nu
    int 21h
    sfarsit:
    mov ah,4ch
    int 21h ; stop program
    END pstart
```

**5**. Program care calculează suma cifrelor unui număr introdus de la tastatură.

```
    ; Programul calculează suma cifrelor unui număr introdus de la
;tastatură
    dosseg
    .model small
    .stack
    .data
```

```
        nr DB 10,10 dup(?)
        rezultat DB 10,10 dup(?)
        mesaj  db 13,10,'Introduceți numărul:$'
        mesaj_suma db 13,10,'Suma cifrelor numărului este: $'
        .code
        pstart:
              mov ax,@data
              mov ds,ax
             mov ah,09          ; aici se afişează mesajul inițial de introducere
              mov dx,offset mesaj       ; a numărului
              int 21h
mov ah,0ah      ; funcția 10(0ah) citeşte un şir de caractere de la
;tastatura intr-o variabilă de memorie
                  mov dx,offset nr
                  int 21h
                  mov si,1
                  mov cl,nr[si]   ; încarc în CL numărul de cifre al numărului
;introdus
                  and cx,00FFh
                  inc cx          ; CX stochează acum ultima poziție din şirul
;de cifre
                  xor ax,ax       ; stocăm rezultatul în AX, pe care îl
;inițializăm cu zero
        următorul_caracter:
                  inc si          ; SI creşte de la începutul şirului spre sfârşit
                  add al,nr[si]
                  sub al,30h      ; scădem codul ASCII al lui zero
                  cmp si,cx       ; în şir se va merge până la poziția cl+1
                  jne următorul_caracter
                  xor si,si       ; SI este indicele din şirul care va conține
;rezultatul
        cifra: ; aici începe afişarea rezultatului din AX
                  mov bx,0ah
                  div bx
                  add dl,30h
                  mov rezultat[si],dl
                  inc si
                  xor dx,dx
                  cmp ax,0
                  jne cifra
                  mov ah,9
```

```
                mov dx,offset mesaj_suma
                int 21h
        caracter:
                dec si
                mov ah,02               ;apelarea funcției 02 pentru afişarea
;unui caracter
                mov dl,rezultat[si]     ;al cărui cod ASCII este în DL
                int 21h
                cmp si,0
                jne caracter
                mov ah,4ch
                int 21h                 ; terminarea programului
        END pstart
```

# 4

# MEMORIA ŞI MAGISTRALELE DE SISTEM

## 4.1 Introducere

După cum am văzut în capitolul 1, memoria şi componenta de conexiune (magistrala de sistem) reprezintă, pe lângă microprocesor şi interfețele de intrare/ieşire, elemente arhitecturale de bază ale unui calculator. Memoria calculatorului este absolut indispensabilă pentru execuția programelor, având în vedere faptul că, pentru a fi executat, un program trebuie încărcat în prealabil în memorie. Una dintre funcțiile importante ale unui sistem de operare pentru calculator îl reprezintă managementul memoriei, adică administrarea întregului spațiu de memorie, alocarea de spațiu în memorie pentru diferite aplicații şi asigurarea faptului că acestea nu interferează între ele.

De regulă, sistemul de operare este alcătuit dintr-o serie de rutine software, dintre care unele asigură controlul operațiilor de intrare/ieşire, oferind suport programelor de aplicații aflate în execuție; aceste rutine *rezidente* trebuie să rămână permanent încărcate în memorie. Spre deosebire de rutinele rezidente, o serie de rutine trebuie încărcate în memorie doar atunci când este nevoie de ele; aceste rutine se numesc rutine *tranziente*. În general, rutinele rezidente ale sistemului de operare sunt încărcate la începutul spațiului de memorie (memoria *joasă – low memory*) existent, de la adresa 0. Restul spațiului de memorie rămas se numeşte *spațiul tranzient* de memorie, unde sunt încărcate rutinele tranziente sau celelalte programe de aplicații.

Necesitatea de a rula mai multe programe în mod concurent* (în aceeaşi perioadă de timp) pentru a mări eficienţa de calcul a sistemului, a condus la creşterea continuă a necesarului de memorie şi, implicit, la dezvoltarea continuă a capacităţilor de memorie.

Dar să vedem în continuare care sunt principalele tipuri de memorie utilizată într-un calculator. Având în vedere că un microprocesor Pentium poate adresa $2^{32}$ locaţii de memorie (4 GB de memorie), memoria maximă ce poate fi instalată pe un calculator echipat cu un astfel de procesor este de 4 GB de memorie. Acest tip de memorie este denumit şi memoria principală sau memoria RAM (Random Access Memory). Am văzut că regiştrii microprocesorului reprezintă locaţii de memorie ultra-rapide, cu un timp de acces de aproximativ 2 ns (nano secunde!). Memoria RAM oferă însă un timp de acces mult mai mare, de aproximativ 60 ns. Din această cauză a apărut un tip intermediar de memorie, memoria *cache*. Memoria cache integrată direct pe cipul microprocesorului se numeşte *memorie cache de nivel 1*, iar memoria cache situată pe placa de bază lângă microprocesor se numeşte memorie *cache de nivel 2* (această memorie cache este ceva mai lentă decât memoria cache de nivel 1 dar şi mai ieftină). O ierarhie a memoriei este prezentată în figura 4.1.

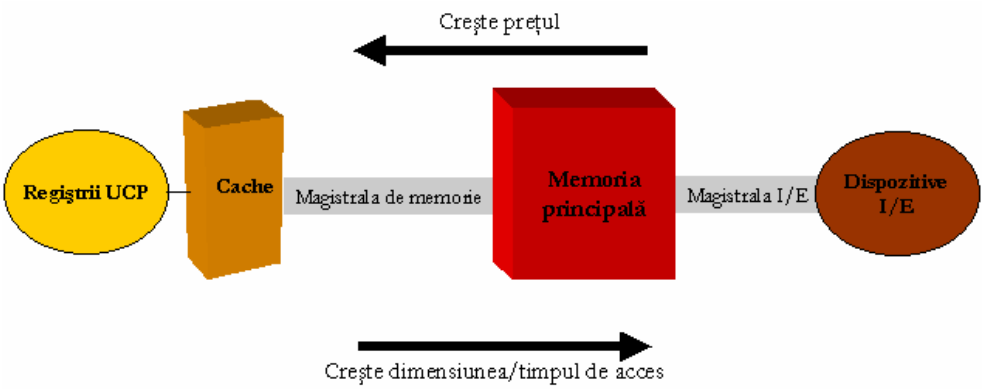



Fig. 4.1 O ierarhie a memoriei

* Este cunoscut faptul că microprocesorul aduce din memorie şi execută o singură instrucţiune într-un ciclu de tact. Dacă procesorul poate executa o singură instrucţiune la un moment dat, desigur că nu poate executa mai multe programe la *un moment dat (simultan)*. Nici un sistem monoprocesor nu poate executa două sau mai multe programe *simultan*. Programele sunt însă rulate în mod *concurent*, aceasta însemnând că sunt rulate în *aceeaşi perioadă de timp.*

Observăm din desenul de mai sus că regiştrii procesorului şi memoria cache reprezintă cel mai rapid tip de memorie (timp de acces mic), dar şi cel mai scump tip de memorie. Analizând tabelul 4.2 putem avea o imagine şi mai precisă legată de tipul de memorie, dimensiunea, viteza, lăţimea de bandă *(bandwidth),* modul de administrare şi backup.

## 4.2 Memoria cache

Să considerăm cazul unui procesor Intel ce rulează la frecvenţa de ceas de 500 Mhz; acest lucru presupune că acesta are nevoie pentru acces la memorie (în medie) de 2 ns pentru a aduce o instrucţiune (etapa fetch) şi de ceva mai mult timp pentru a aduce datele din memorie. Memoria de tip EDO RAM are un timp de răspuns de aproximativ 60 ns, de unde rezultă o mare disproporţie între "viteza de mişcare" a procesorului şi cea a memoriei RAM. Din acest punct de vedere, dacă microprocesorul ar dispune doar de memorie RAM, ar lucra de 30 de ori mai încet! Din fericire, pentru ca procesorul să nu lucreze mai încet din cauza pierderii de timp cu accesul la memoria lentă RAM, a fost găsită soluţia: memoria cache.

Tabelul 4.2

**Principalele tipuri de memorie - caracteristici**

| | Regiştrii UCP | Cache | Memoria principală | Dispozitive I/E |
|---|---|---|---|---|
| **Dimensiune** | 200 B | 64 KB | 32 MB | 2 GB |
| **Viteza** | 5 ns | 10 ns | 100 ns | 5 ms |
| **Dimensiuni maxime** | 1 KB (CMOS sau BiCMOS) | 4 MB SRAM | 4 GB DRAM | > 1 GB |
| **Bandwith (MB/sec)** | 4000-32000 | 800-5000 | 400-2000 | 4-32 |
| **Administrare** | Compilator | Hardware | SO | SO/utilizator |
| **Backup** | Cache | Mem principală | Disc | Banda magnetică |

Memoria de tip cache este mai rapidă decât memoria RAM obişnuită datorită faptului că este construită din cipuri de memorie de tip SRAM (Static RAM) ce operează la viteze superioare decât memoria de tip DRAM (Dynamic RAM). În cazul ideal, viteza de acces a memoriei cache ar trebui să fie comparabilă cu cea a vitezei de ceas a microprocesorului, permiţând astfel acestuia să acceseze instrucţiunile şi datele fără nici o întârziere. Majoritatea unităţilor UCP sunt mai rapide decât viteza de acces a memoriei şi, de aceea, trebuie să facă pauze pentru a „aştepta" echipamentele mai încete (memoria sau dispozitivele periferice, de exemplu) să reacţioneze.

În acest caz, există un controlor al magistralei de sistem care generează stări de aşteptare (*wait states*) unității centrale de procesare în execuția unui ciclu *fetch-decode-execute* obişnuit. Memoria cache împreună cu unitatea de control a memoriei cache (denumită, de regulă, CCU – Cache Controller Unit) este interpusă între UCP şi memoria principală (vezi figura 4.3), de unde poate prelua toate cererile de acces la memorie ce vin de la UCP. Scopul principal este acela de a menține în memoria rapidă cache porțiunile de memorie active cu date şi instrucțiuni. Toate operațiile de citire/scriere inițiate de UCP sunt direcționate către memoria cache în speranța unui răspuns mai rapid.

Cum acționează în linii mari memoria cache? Unitatea de control a memoriei cache (UCC – Unitate de Control Cache) verifică adresa de memorie cerută de UCP pentru a vedea dacă aceasta se află în memoria cache. Dacă este adevărat acest lucru, UCP poate accesa valoare de memorie imediat, în caz contrar controlorul UCC trebuie să aducă respectiva valoare din memoria principală, de unde rezultă o întârziere în procesare. Acest procedeu are ca rezultat mărirea vitezei de procesare, mai ales în cazul în care procesorul are de executat un program alcătuit din instrucțiuni de ciclare aflate în întregime în micul spațiu de memorie cache. Şi în cazul instrucțiunilor secvențiale se constată o mărire a vitezei de execuție, mai ales dacă controller-ul de memorie cache are abilitatea să „ghicească" înainte instrucțiunile ce vor fi executate şi să le încarce în memoria cache.

Atunci când o valoare cerută de microprocesor este găsită în memoria cache, operația respectivă se numeşte *cache hit*, în timp ce în caz contrar avem de-a face cu un *cache miss*. În acest din urmă caz un bloc de dimensiune fixă de date ce conține valorile necesare sunt extrase din memoria principală şi introduse în memoria cache. De regulă, în funcționarea memoriei cache fenomenul de *cache miss* apare destul de rar, datorită unei reguli de „localizare temporală" a datelor în calculator, regulă care spune că ultimile date utilizate au o probabilitate foarte mare să fie accesate din nou în curând, deci aceste date sunt plasate în memoria cache, de unde pot fi accesate foarte rapid. Timpul de răspuns pentru un astfel de *cache miss* depinde atât de latența memoriei cât şi de lățimea de bandă a magistralei – ce determină timpul în care va fi citit întregul bloc de date. Un *cache miss* administrat hardware va determina de regulă o pauză în funcționarea UCP, până în momentul în care sunt disponibile datele.

Unele compilatoare administrează acest proces prin inserarea unor instrucțiuni speciale care stimulează controlorul de memorie cache să

anticipeze următoarele instrucţiuni ce vor fi executate. O dată cu introducerea unor noi valori în memoria cache, trebuie făcută o decizie legată de scoaterea altor valori din aceeaşi memorie, pentru a elibera spaţiul. Poate apărea astfel o nouă întârziere rezultată din scrierea valorii înapoi în memoria principală a tuturor valorilor care au fost modificate în timpul stocării lor în memoria cache. Diferenţa între memoria cache de nivel 1 (*level 1 cache*) şi memoria cache de nivel 2 (*level 2 cache*) este aceea a localizării: dacă memoria cache de nivel 1 (sau memorie *cache primară*) este situată direct pe cipul microprocesorului, memoria cache de nivel 2 (denumită şi memorie *cache secundară*) este situată de regulă pe placa de bază, cât mai aproape de microprocesor.

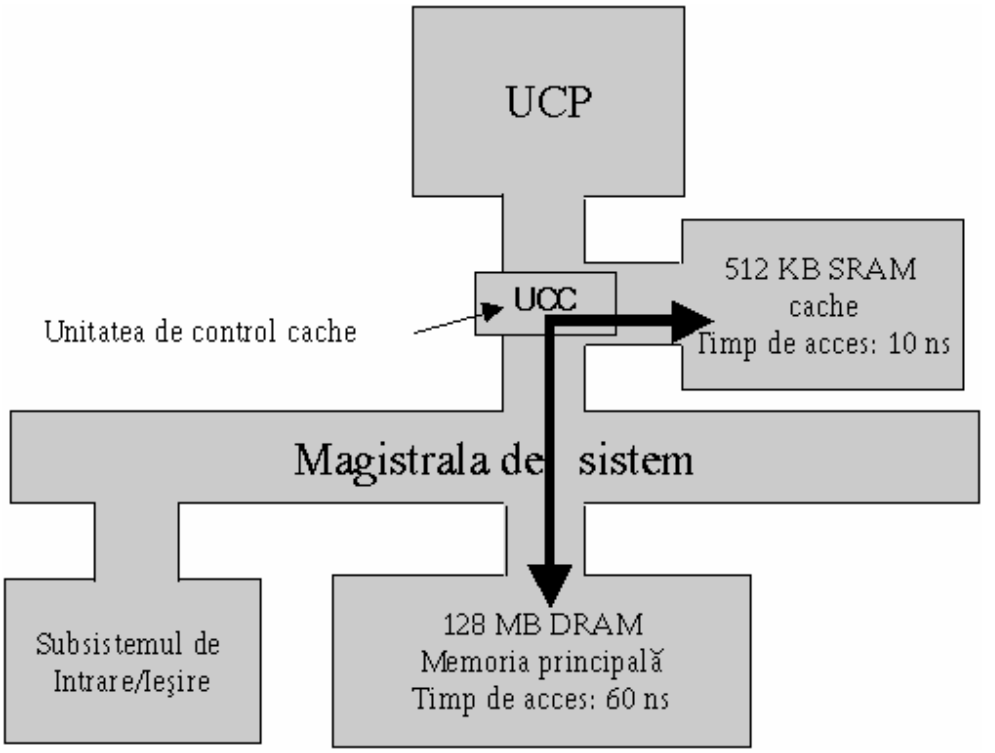



Fig. 4.3 Memoria şi controlorul de memorie cache

Datorită plasării direct pe cipul microprocesorului, memoria cache de nivel 1 are dimensiuni mai mici decât cea de nivel 2. De exemplu, la Pentium, memoria cache de nivel 1 are 16 KB, împărţită în două jumătăţi, 8 KB pentru instrucţiuni şi 8 KB pentru date. Această diferenţiere între instrucţiuni şi date (este vorba despre aşa numită ***„Arhitectură Harvard”***)

permite aducerea simultană din memorie a instrucţiunilor şi accesul la variabile dacă magistrala de sistem este împărţită, de asemenea, în magistrală de date şi magistrală de instrucţiuni. Cu ajutorul unui decodor pipeline, magistralele duale pot preveni blocările apărute în momentul în care trebuie transferate mai multe valori în acelaşi timp. Acest lucru este posibil pentru memoria cache primară, nu şi pentru memoria cache secundară.

Memoria cache secundară are dimensiuni mai mari (de regulă aceasta are valori de 256 KB, 512 KB, 1MB sau chiar 2 MB), fiind localizată pe placa de bază în apropierea procesorului, dar are şi viteze mai scăzute de operare, datorită întârzierilor apărute ca urmare a semnalelor trimise de-a lungul magistralei de sistem. Pentru a îmbunătăţi această situaţie, la Pentium II Intel a înglobat memoria cache de nivel 2 în cadrul cartuşului denumit Slot1/Slot2, la un loc cu microprocesorul. Acest cartuş include o magistrală locală dedicată procesorului şi memoriei cache de nivel 2, ce operează la 200 MHz, adică dublu faţă de magistrala de sistem a plăcii de bază ce rulează la 100 MHz.

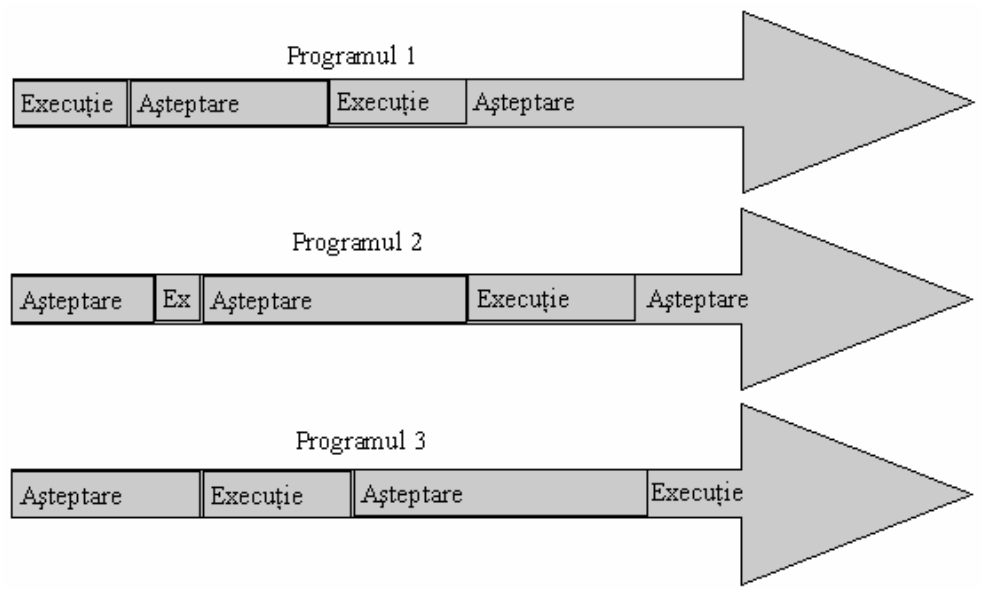



Fig. 4.4 Execuţia concurenţială a mai multor programe încărcate în memorie

### 4.3 Regiuni şi partiţii de memorie

Sistemele de operare multitasking presupun execuţia concurenţială (în acelaşi *interval* de timp) a mai multor programe. Majoritatea sistemelor de operare de astăzi sunt sisteme de operare multitasking (doar vechiul sistem de operare DOS a rămas un exemplu de sistem monotasking) în care mai multe programe sunt încărcate în memorie şi rulate în aceeaşi perioadă de timp (figura 4.4). Un program poate fi într-una din cele două stări: execuţie sau aşteptare. Observăm din desen că în timp ce programul 1 este în stare de aşteptare (poate aştepta venirea datelor din memorie), procesorul începe să ruleze programul 2, iar în timp ce programul 2 este în stare de aşteptare, procesorul îşi concentrează atenţia asupra prorgamului 3. De regulă acest procedeu de execuţie concurenţială este utilizat de toate sistemele de operare multitasking, lucru care însă implică un management al memoriei adecvat.

Cea mai simplă abordare a administrării memoriei pentru mai multe programe ce rulează concurenţial este administrarea memoriei folosind partiţii fixe de memorie, caz în care spaţiul de memorie disponibil se împarte în partiţii de dimensiune fixă, fiecare dintre aceste partiţii stocând câte un program. Dimensiunea partiţiilor este stabilită o dată cu iniţializarea sistemului de operare , astfel încât decizia de alocare a memoriei pentru un program se face înainte ca acel program să aibă efectiv nevoie de acel spaţiu de memorie. Din cauză că dimensiunea alocată trebuie să fie suficient de mare pentru ca cel mai mare dintre programe să poată încăpea în memorie, administrarea memoriei folosind partiţii fixe face risipă de spaţiu de memorie, dar are avantajul că este o metodă foarte simplă.

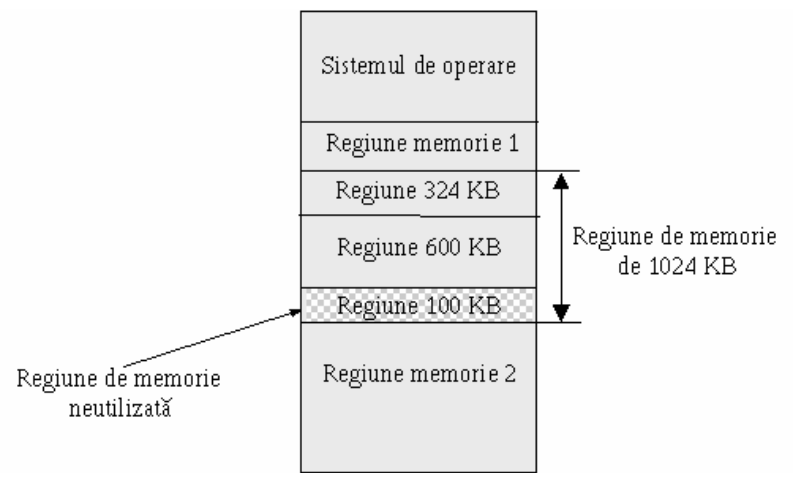



Fig. 4.5 Fragmentarea memoriei

O altă metodă de administrare a memoriei o reprezintă administrarea dinamică a memoriei, caz în care spațiul tranzient de memorie este tratat ca un întreg. Atunci când sistemul de operare decide să încarce un anumit program în memorie, se alocă o regiune de memorie liberă din acest întreg, suficientă pentru ca acel program să ruleze. Din cauză că programul respectiv primeşte doar spațiul de care are nevoie, risipa de memorie este mult mai mică. Administrarea dinamică a memoriei nu rezolvă în întregime problema spațiului de memorie neutilizat; să presupunem că un program de 1024 KB tocmai şi-a încheiat execuția şi sistemul de operare nu găseşte nici un alt program de 1024 KB pentru încărcare în memorie, în schimb încarcă două programe, unul de 324 KB şi unul de 600 KB (figura 4.5). În acest caz rămân, totuşi, 100 KB de memorie nefolosită, dacă nu există programe mai mici pentru a fi încărcate în această zonă. Apar astfel mici zone de memorie neutilizate, creând o problemă de *fragmentare*.

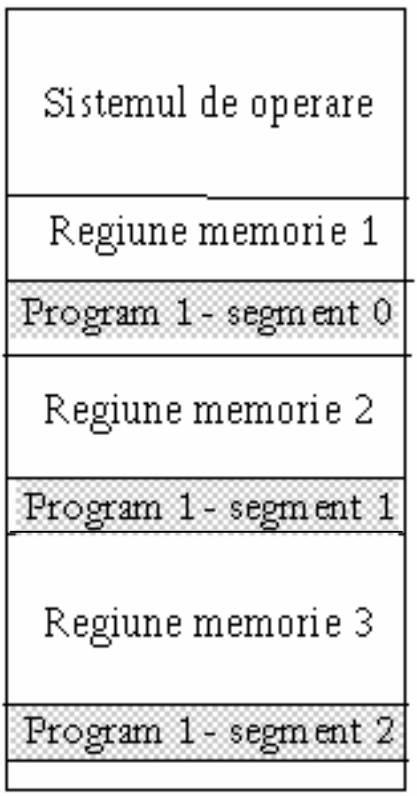



Fig. 4.6 Segmentarea memoriei

### *4.3.1 Segmentarea*

Motivul pentru care apare problema fragmentării memoriei este acela că atât administrarea memoriei folosind partiţii fixe, cât şi administrarea dinamică a memoriei presupun că programul trebuie să fie încărcat în zone continue (locaţii adiacente) de memorie. Prin segmentare, un program poate fi împărţit în segmente independente şi stocat în memorie în zone necontinue (figura 4.6).

Segmentarea memoriei implică un pas în plus în mecanismul de translatare a adreselor de memorie. Atunci când un program este încărcat în memorie, sistemul de operare construieşte o tabelă de segmente în care sunt memorate adresele absolute pentru fiecare segment de program (pentru fiecare program activ există un tabel de segmente de memorie). În continuare, când sistemul de operare lansează în execuţie un program, va încărca adresa din tabelul de segmente într-un registru special.

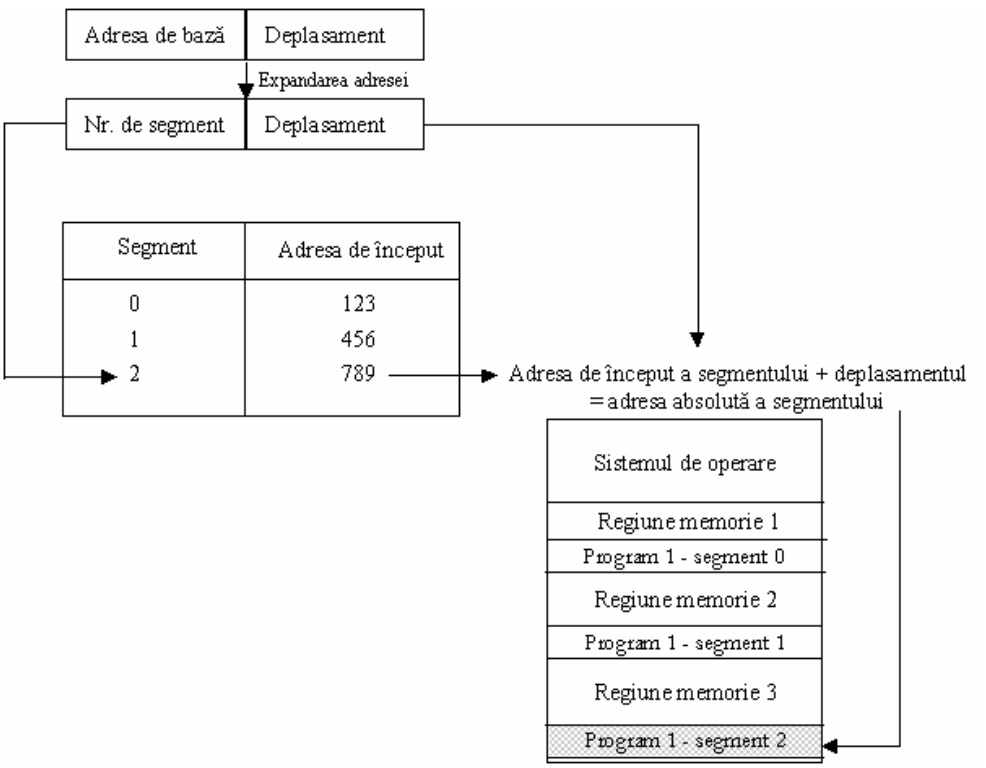



Fig. 4.7 Procedeul de translatare dinamică a adreselor

În timpul execuției programului, adresele trebuie translatate din adrese relative în adrese absolute deoarce programatorii pot scrie acelaşi cod de program iar compilatoarele vor genera adresa completă, de tipul *adresa de bază + deplasament*.

După ce instrucțiunea este adusă din memorie (etapa *fetch*), unitatea de control a instrucțiunilor dezvoltă fiecare adresă a operanzilor prin adăugarea la registrul de bază (ce conține adresa de bază) a adresei de deplasament. În mod normal, acest lucru ne conduce la adresa absolută. În cazul segmentării, adresa expandată va conține o adresă compusă din două părți: un număr de segment şi un deplasament (figura 4.7).

Conversia hardware a adresei segment/deplasament în adresă absolută se face în felul următor:

- Se verifică registrul special pentru a găsi tabela de segmente a programului;
- Din adresa expandată se extrage numărul de segment;
- Numărul de segment se foloseşte pentru a căuta tabela de segmente a programului;
- Se găseşte adresa absolută a segmentului;
- Se adaugă adresa de deplasament la adresa absolută a segmentului pentru a obține adresa absolută.

Acest proces de translatare se numeşte *translatarea dinamică a adreselor*.

### *4.3.2 Paginarea*

Segmentele de program încărcate în memorie pot avea dimensiuni variabile, de aceea s-a recurs la soluția *paginării*, adică a împărțirii unui program în *pagini* de lungime fixă. Dimensiunile paginilor sunt de dimensiune mică (2KB sau 4 KB, spre exemplu) pentru a eficientiza încărcarea acestora în memorie. Ca şi în cazul segmentelor, paginile de program pot fi încărcate în zone ne-continue de memorie. În acest caz, adresele sunt formate din două părți: un număr de pagină în partea cea mai semnificativă şi un deplasament pe pozițiile biților cel mai puțin semnificativi. De asemenea, adresele sunt translatate în mod dinamic în timpul execuției programului. Atunci când o instrucțiune este adusă din memorie, adresele de tipul *bază+deplasament* sunt expandate la adrese

absolute în hardware, apoi se caută adresa de început a paginii respective din tabela de pagini (asemănătoare cu tabela de segmente) şi apoi se adaugă deplasamentul, exact ca în cazul segmentării (figura 4.8).

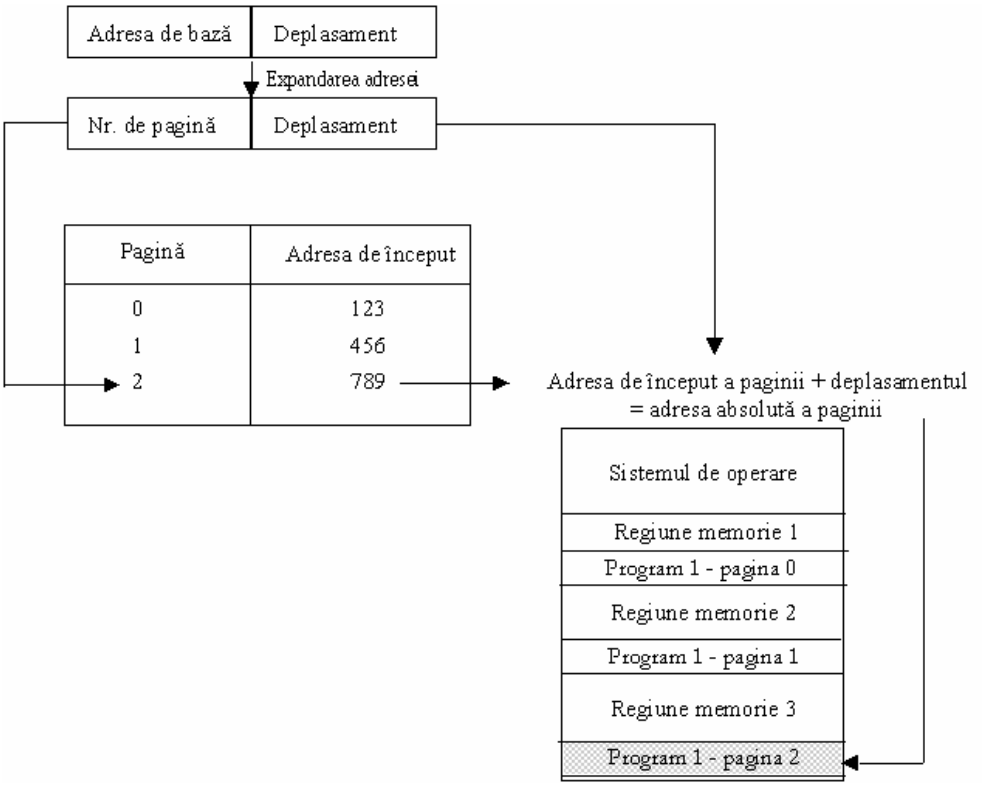



Fig. 4.8 Translatarea adresei în cazul paginării

### *4.3.3 Segmentare şi paginare*

Combinând segmentarea cu paginarea, rezultă un procedeu în care adresele de memorie sunt compuse dintr-un număr de segment, un număr de pagină în cadrul acelui segment şi un deplasament în cadrul paginii. După ce unitatea de control expandează adresa relativă, începe translatarea dinamică a adresei. În primul rând se extrage adresa segmentului programului, apoi din tabela de segmente se extrage adresa tabelei de pagini pentru segmentul respectiv. În final, din tabela de pagini a segmentului se obține adresa reală care adunată cu deplasamentul formează adresa absolută (figura 4.9).

De regulă, administrarea memoriei este un proces complicat de care trebuie să aibă grijă sistemul de operare. Conținutul memoriei RAM se

schimbă permanent; deoarece există multe programe care partajează această memorie, este posibil uneori ca unul dintre programe să altereze conţinutul de memorie alocat altui program, astfel încât programele active din memorie trebuie să fie protejate unele de altele. Aceasta este sarcina sistemului de operare, astfel încât dacă un program încearcă să modifice sau pur şi simplu să acceseze spaţiul de memorie alocat altui program, mecanismul de protecţie a memoriei sistemului de operare va interveni şi poate întrerupe acel program.

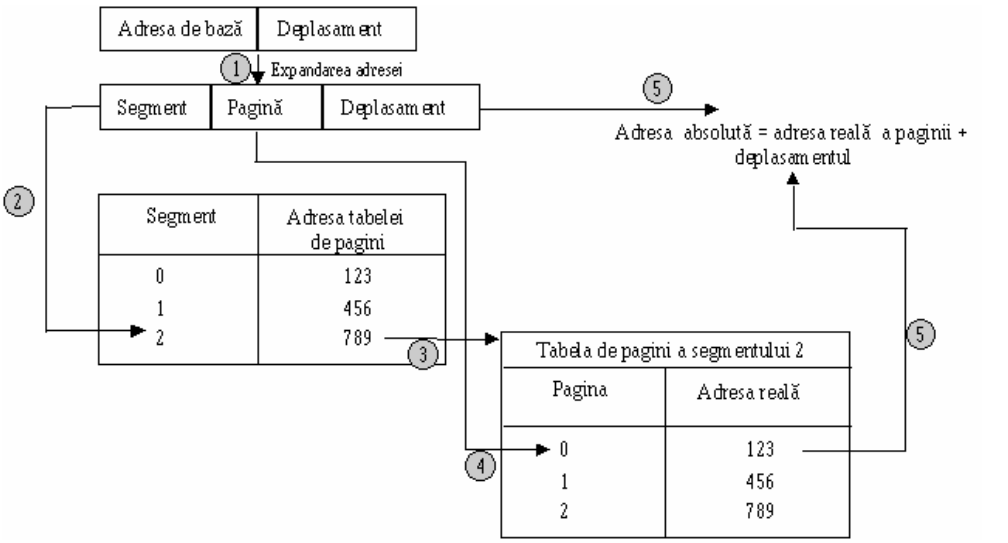



Fig. 4.9 Translatarea adreselor în cazul paginării *şi* segmentării

## 4.4 Modalităţi de stocare a datelor în memorie

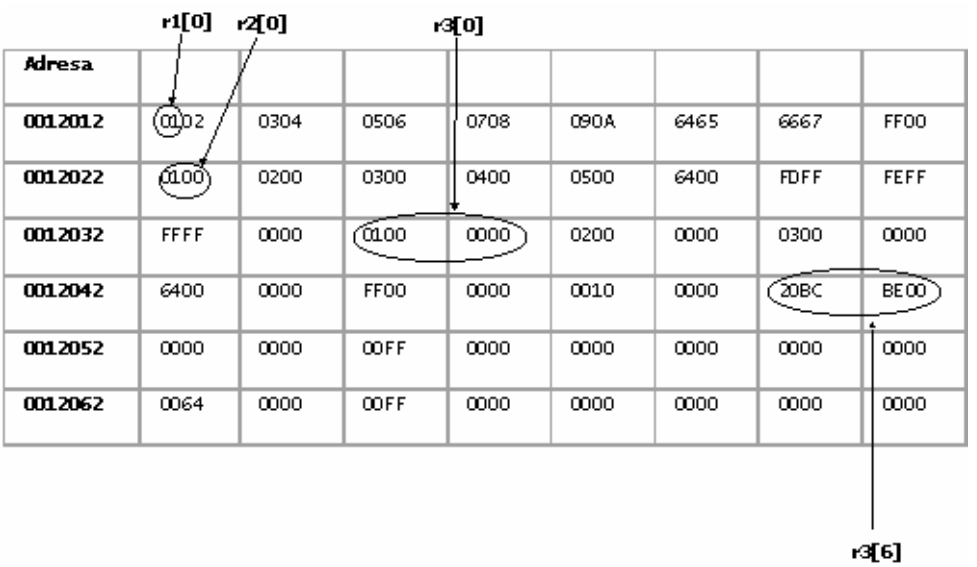
### *Stocarea Little Endian vs. Big Endian*

Modalitatea de aranjare a datelor (reprezentate pe mai mulţi octeţi) în memoria calculatorului nu este aceeaşi pentru toate tipurile de microprocesoare. De fapt, cei doi „titani" ai producţiei de cipuri microprocesoare, Intel şi Motorola, au stabilit convenţii diferite de reprezentare a datelor.

Iniţial, Intel a stabilit o convenţie de reprezentare a datelor de dimensiune de 16 biţi (2 octeţi) prin împărţirea în doi octeţi şi stocarea în memorie întâi a octetului cel mai puţin semnificativ (denumit şi

*CMPSO – cel mai puţin semnificativ octet*) şi apoi a octetului *CMSO – cel mai semnificativ octet*. Această convenţie a luat numele de *Little-Endian*.

Mai târziu, Motorola a dorit să schimbe această modalitate de stocare a datelor pe 16 biţi, raţionând în mod invers: se stochează mai întâi octetul cel mai semnificativ şi după aceea octetul mai puţin semnificativ. Această nouă convenţie a fost denumită *Big-Endian*. Această convenţie are avantajul că reprezentarea datelor în memorie este identică, de fapt, cu aceea a scrierii obişnuite a numerelor, numită şi ordinea de tipărire (*„print order”*).



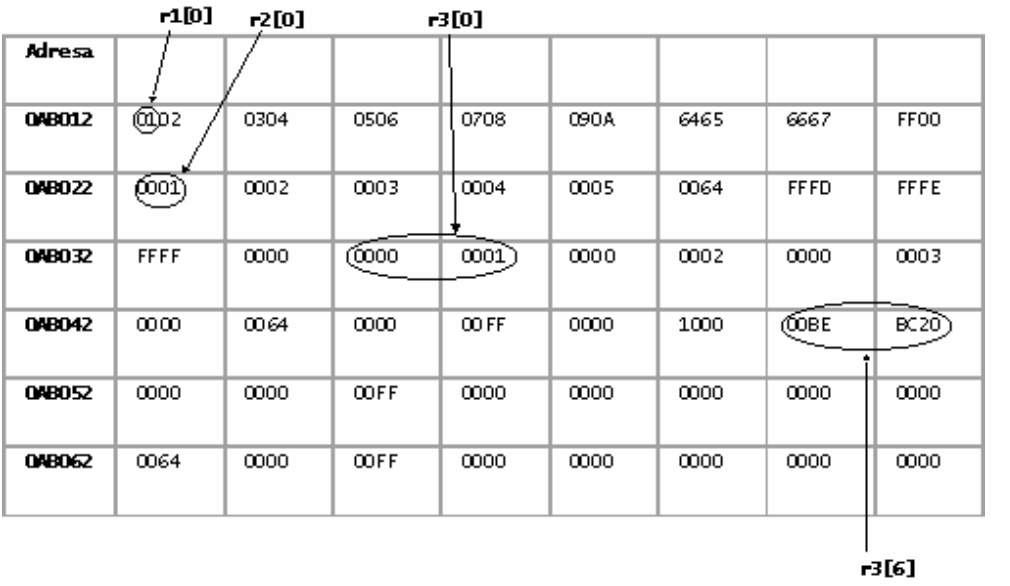

| Adresa | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 0012012 | 0102 | 0304 | 0506 | 0708 | 090A | 6465 | 6667 | FF00 |
| 0012022 | 0100 | 0200 | 0300 | 0400 | 0500 | 6400 | FDFF | FEFF |
| 0012032 | FFFF | 0000 | 0100 | 0000 | 0200 | 0000 | 0300 | 0000 |
| 0012042 | 6400 | 0000 | FF00 | 0000 | 0010 | 0000 | 20BC | BE00 |
| 0012052 | 0000 | 0000 | 00FF | 0000 | 0000 | 0000 | 0000 | 0000 |
| 0012062 | 0064 | 0000 | 00FF | 0000 | 0000 | 0000 | 0000 | 0000 |

Fig. 4.10 Reprezentarea Little-Endian la Intel Pentium

Cele două convenţii diferite de stocare a datelor sunt prezentate în figurile 4.10 şi 4.11. Figura 4.10 prezintă conţinutul memoriei (aşa cum apare înfăţişat de un program depanator) în cazul unui procesor Intel Pentium. Porţiunea de memorie vizualizată conţine trei vectori de întregi cu date de tip ***byte*** (reprezentate pe un singur octet), ***word*** (reprezentate pe doi octeţi) şi ***int*** (reprezentate pe patru octeţi). Declaraţiile în C corespunzătoare celor trei vectori sunt următoarele:

```
unsigned char r1[]={1,2,3,4,5,6,7,8,9,10,100,101,
102,103,255}
unsigned short r2[]={1,2,3,4,5,100,65533,65534,65535}
unsigned int r3[]={1,2,3,100,255,4096,12500000}
```

În figura 4.11 este prezentat conținutul memoriei pentru aceeaşi trei vectori de întregi de data aceasta în cazul unui microprocesor Motorola 68030. În cele două figuri, fiecare octet apare în reprezentare hexazecimală (ca o pereche de două cifre hexa, având în vedere că o cifră hexa se reprezintă pe 4 biți). Drept urmare, valoarea 02 este în binar 0000 0010 iar AF este, în binar, 1010 1111.




| Adresa | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 0AB012 | 0102 | 0304 | 0506 | 0708 | 090A | 6465 | 6667 | FF00 |
| 0AB022 | 0001 | 0002 | 0003 | 0004 | 0005 | 0064 | FFFD | FFFE |
| 0AB032 | FFFF | 0000 | 0000 | 0001 | 0000 | 0002 | 0000 | 0003 |
| 0AB042 | 0000 | 0064 | 0000 | 00FF | 0000 | 1000 | 00BE | BC20 |
| 0AB052 | 0000 | 0000 | 00FF | 0000 | 0000 | 0000 | 0000 | 0000 |
| 0AB062 | 0064 | 0000 | 00FF | 0000 | 0000 | 0000 | 0000 | 0000 |

Fig. 4.11 Reprezentarea Big-Endian la Motorola 68030

Din analiza celor două tabele observăm că întregii reprezentați pe un octet (tipul ***char***) au aceeaşi reprezentare în ambele cazuri (Intel şi Motorola). Întregii pe 16 biți (2 octeți – tipul ***short***) au cei doi octeți inversați, la fel ca şi în cazul întregilor pe 32 de biți (4 octeți – tipul ***int***). În concluzie, observăm că dacă toți vectorii ar fi de tipul ***unsigned char*** (pe un singur octet), nu ar fi apărut nici o diferență sau problemă de ordonare. Citirea unui fişier stocat prin folosirea convenției *Little-Endian* pe un calculator ce utilizează *Big-Endian* va fi însă imposibilă.

De asemenea, atunci când se transferă fişiere ce conțin date numerice de pe un PC pe un calculator UNIX poate apărea problema inter-schimbării octeților; în acest sens, pe UNIX există un program intitulat ***dd*** care poate face conversia între formatele *Little-Endian* şi *Big-Endian*. De exemplu, pentru a transfera fişiere de pe un calculator PC pe un calculator Sun, ele trebuiesc convertite. Sun utilizează convenția de la Motorola (*Big-Endian*) în timp ce Microsoft foloseşte convenția Intel (*Little-Endian*).

O altă diferență apare, de regulă, la transferul unui fişier text de pe un calculator UNIX pe un PC sau invers. Acest lucru se întâmplă datorită caracterului utilizat pentru marcarea sfârşitului de linie. Aplicațiile de pe calculatoarele PC folosesc caracterul ^M (CR – Carriage Return – Retur de car) în timp ce aplicațiile de pe UNIX utilizează caracterul ^J (LF – Line Feed – Linie nouă).

În figura 4.12 este prezentat un alt exemplu pe 64 de biți (8 octeți). Dacă la Motorola reprezentarea în memorie este reprezentarea „print order" (cazul a), la Intel cei 8 octeți sunt scrişi în memorie în ordine inversă decât la Motorola (cazul b).

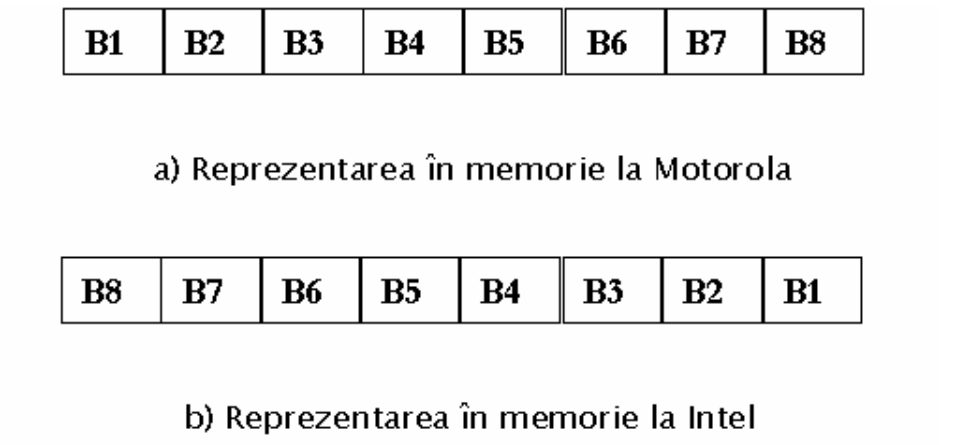



Fig. 4.12 Reprezentări Big-Endian şi Little Endian pe 64 de biți

## 4.5 Memoria virtuală

### *4.5.1 Introducere*

Memoria virtuală semnifică (figura 4.13) separarea conceptuală a memoriei logice disponibile pentru aplicații față de memoria fizică (reală). Prin intermediul acestui concept putem avea o memorie virtuală (disponibilă pentru aplicații) de dimensiuni mari chiar cu o memorie fizică de dimensiuni reduse. Conceptul de memorie virtuală îşi are originile de pe vremea în care memoria calculatorului era foarte scumpă şi nu prea rapidă. Mecanismul memoriei virtuale oferea o extindere convenabilă a memoriei prin utilizarea hard-diskului pe post de memorie. Spațiul de pe hard-disk rezervat în acest scop este cunoscut sub denumirea de spațiu *swap* şi este administrat de către

sistemul de operare, în timp ce programatorii nu trebuie să se preocupe de acest lucru, el fiind administrat de către serviciile de sistem pentru administrarea memoriei virtuale. Fie că este vorba despre una dintre variantele de Windows 95/98, NT, 2000, XP sau despre UNIX şi Linux, toate aceste sisteme de operare implementează mecanismul memoriei virtuale. În continuare vom prezenta mai în detaliu cum se realizează acest proces.

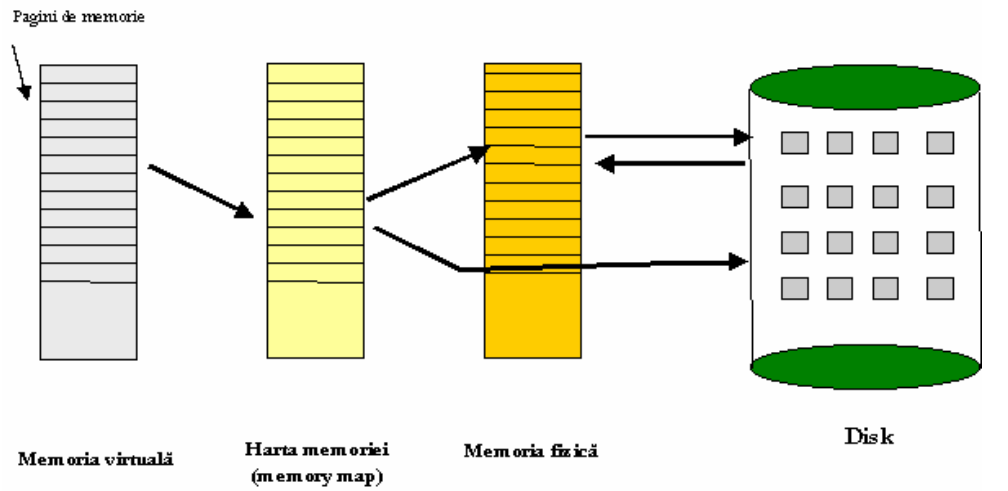



Fig. 4.13 Ilustrarea conceptului de memorie virtuală

Înainte ca această tehnică să devină foarte cunoscută şi larg implementată, singura opțiune pentru programatori (atunci când programele deveneau prea mari ca să încapă în memoria principală) era aceea de a împărți programele în mai multe segmente legate între ele. Primul dintre aceste segmente era încărcat în memorie de pe disc şi îşi începea execuția, iar la terminare comanda încărcarea celui de-al doilea segment în memorie, moment în care cel de-al doilea segment prelua controlul. Procedeul se continua până când toate segmentele programului erau executate.

În zilele noastre însă, discurile magnetice (hard-diskurile) sunt componente periferice de bază ale calculatoarelor care influențează modalitatea de procesare a datelor de către aplicații. Majoritatea sistemelor personale şi a sistemelor de calcul profesionale utilizează hard-diskuri pentru stocarea datelor, acest lucru făcându-se la un avantajos raport preț/performanță, sau, mai bine spus, preț/capacitate de stocare. Cu toate dezvoltările rapide din domeniul memoriilor bazate pe semiconductori,

producătorii de hard-diskuri au reuşit întotdeauna să țină pasul şi să fie aibă prețuri fără concurență când este vorba despre prețul pe octet stocat.

După cum îi spune şi numele, *memoria virtuală* nu este reală; pentru programator sau pentru utilizator, această memorie virtuală acționează precum memoria obişnuită, dar nu este memorie reală. Memoria virtuală este formată, de fapt, atât din memorie reală, obişnuită, cât şi dintr-o porțiune de spațiu de pe hard-disk (din *spațiul de swap*). După cum spuneam, sistemele de operare moderne ce înglobează sisteme de administrare a memoriei virtuale permit programatorilor să scrie programe ca şi cum aceste programe sunt singurele ce rulează în sistem şi nu interacționează cu nici un alt program.

Acest lucru de o importanță deosebită a fost obținut pentru prima dată de sistemele de operare multiuser (ce permit folosirea simultană a sistemului de către mai mulți utilizatori) care interschimbau programe întregi în/din memorie după cum era necesar. O astfel de abordare greoaie însă ducea la o utilizare total ineficientă a resurselor şi, în plus, era mare consumatoare de timp. Următoarea abordare a condus la împărțirea programelor în segmente, ceva într-un mod similar cu segmentele de cod, date şi stivă pe care le întâlnim într-un program scris în limbaj de asamblare pentru procesoarele din familia Intel x86. Posibilitatea interschimbării doar a segmentelor şi nu a întregului program în/din memorie a redus consumul de timp şi de resurse, menținând iluzia de "singur utilizator al sistemului". Segmentele de program pot fi amplasate oriunde în memoria calculatorului, având în vedere că acestea sunt localizate prin intermediul regiştrilor de segment. Aceşti regiştri de segment speciali sunt astfel setați încât să facă referire la adresa primei locații a unui segment atunci când segmentul respectiv este încărcat în memorie. Din punct de vedere al securității aplicațiilor acest model este eficient, deoarece implementează în hardware o schemă de securitate a adresării pe bază de segmente de memorie. Datorită însă a dimensiunii variabile a segmentelor de memorie, administratorul de memorie are dificultăți în reutilizarea eficientă a memoriei (apar problemele de fragmentare despre care am mai discutat). În acest sens, pentru a elimina problema apărută în cazul segmentării, schemele eficiente de administrare a memoriei virtuale utilizează mecanismul paginării, cu pagini de memorie de lungime fixă (2 KB sau 4 KB).

O dată cu dezvoltarea crescândă a microprocesoarelor şi scăderea cu mult a prețului memoriei RAM, calculatoarele personale puteau renunța la conceptul de memorie virtuală, dar, oarecum surprinzător, memoria virtuală este „prezentă" pe toate calculatoarele existente astăzi, de la simple

calculatoare personale până la supercomputere. Această situaţie este parţial din cauza caracteristicilor de protecţie şi securitate pe care sistemele de operare ce utilizează memoria virtuală le oferă. Pe lângă administrarea interschimbului de pagini între memoria reală şi spaţiul de swap de pe hard-disk, schema de segmentare a memoriei oferă şi izolarea proceselor şi datelor asociate, deci protecţie.

Datorită faptului că memoria virtuală permite programatorilor să dezvolte programe fără grija ca acestea să încapă în memoria principală, au fost dezvoltate sisteme mai mari, în special în domenii precum aplicaţiile grafice şi comunicaţii. De asemenea, masive de date de dimensiuni enorme pot fi declarate, fără să se ţină seama de dimensiunile memoriei RAM, fapt ce a condus la dezvoltarea sistemelor de gestiune a bazelor de date. Un sistem de administrare a memoriei virtuale modern, cum ar fi cel al sistemelor de operare UNIX sau Windows NT acţionează prin împărţirea memoriei principale în *cadre (frames)* de lungime fixă, tipic de 4 KB.

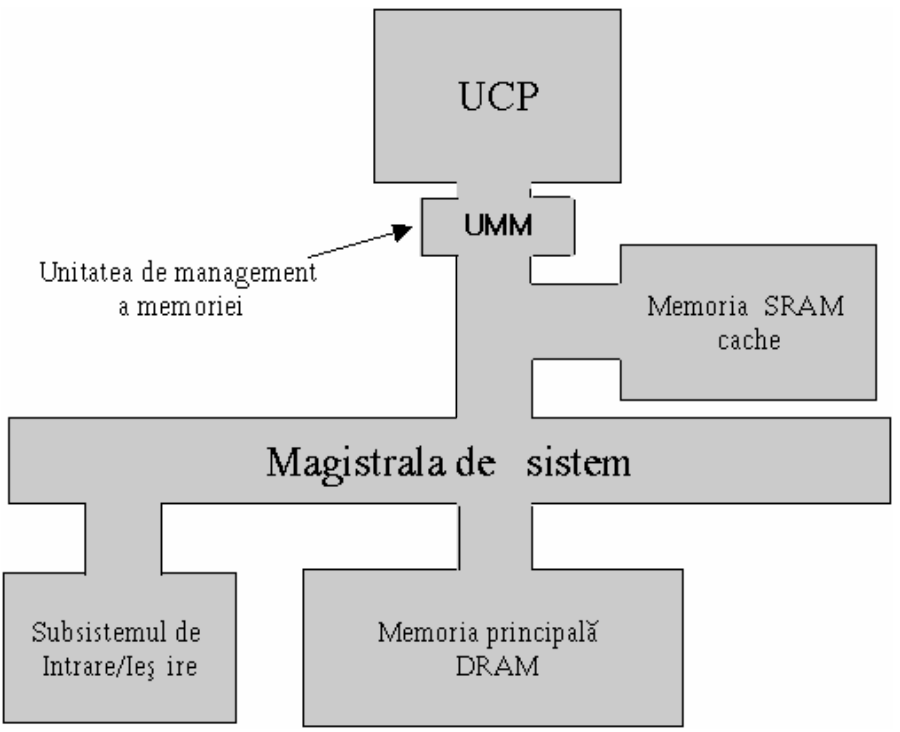



Fig. 4.14 Unitatea de management a memoriei este situată între UCP şi memoria principală

Programul executabil este divizat în acelaşi mod în bucăţi formate din astfel de cadre, denumite *pagini*. Procesorul foloseşte adrese virtuale ce sunt translatate în adrese fizice ce accesează memoria principală. Acest procedeu se numeşte *maparea adreselor* sau *translatarea adreselor* (translatare asemănătoare cu cea de la procedurile de segmentare şi paginare a memoriei). Atunci când un program trebuie să ruleze, nu sunt încărcate toate paginile în memorie, ci doar cele necesare pentru începutul execuţiei programului. Celelalte pagini sunt copiate în spaţiul de swap de pe hard-disk dedicat memoriei virtuale. Procedeul de copiere a unei pagini din spaţiul de swap de pe hard-disk şi de eliberare a unui cadru din memorie poartă numele de *swapping* (interschimbare). În cadrul unui program se poate face o referire la o variabilă sub forma unei adrese logice pe 32 de biţi. Într-un sistem ce utilizează pagini de memorie de 4 KB, cei mai puţin semnificativi 12 biţi reprezintă adresa relativ la pagină, în timp cei 20 de biţi mai semnificativi reprezintă numărul paginii. Tabela de pagini asigură prin mecanismul paginării deducerea adresei fizice conform mecanismului paginării descris anterior.

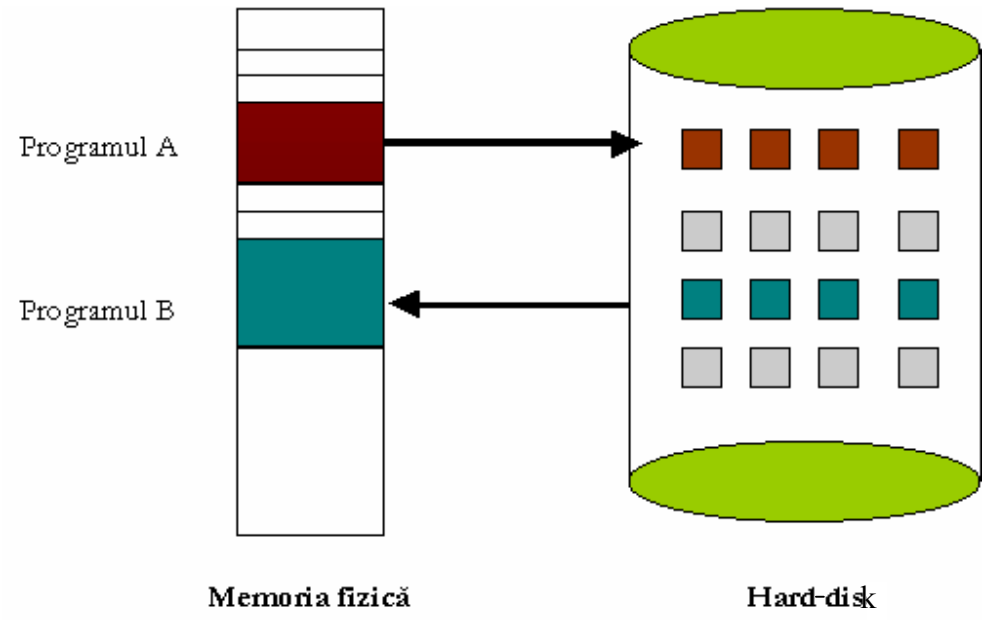



Fig. 4.15 Memoria virtuală – cererea de pagini

Cu toate că nu toate programele pot fi încărcate în acelaşi spaţiu de adrese, mecanismul memoriei virtuale dă iluzia programatorilor că este posibil acest lucru, având în vedere că programele sursă compilate să ruleze

multitasking presupun că adresa de început a codului este totdeauna 0000. Este însă evident că această adresă nu este adresa fizică reală a memoriei, ci doar fenomenul de stabilire a corespondenței între adresa logică şi cea fizică rezolvă această problemă prin repoziționarea paginilor de către sistemul de operare. Din cauza faptului că atât poziția cât şi secvența paginilor de program nu pot fi precizate înainte ca ele să fie încărcate în cadrele de memorie (la momentul compilării şi editării de legături), sistemul de operare foloseşte un translator de adrese. Acest translator se numeşte *Unitatea de management a memoriei* şi este poziționat între UCP şi memorie, incluzând memoria cache (figura 4.14).

### *4.5.2 Cererea de pagini*

Pe parcursul execuției programului, va apărea un moment în care ciclul *fetch-decode-execute* va necesita o instrucțiune dintr-o pagină de memorie. Acest proces de cerere se numeşte chiar *cerere de pagină (demand paging* – figura 4.15). Sunt posibile două cazuri:

**a)** Pagina solicitată este rezidentă în memorie. În acest caz nu este nici o problemă, instrucțiunea respectivă este executată.

**b)** Pagina solicitată nu se află în memorie (acest fenomen poartă numele de *page-fault*). În momentul apariției unei *page-fault*, se inițiază o întrerupere către sistemul de operare care intervine pentru a încărca pagina respectivă din spațiul de swap. De asemenea, dacă nu există nici un cadru liber în memoria principală, mecanismul de administrare a memoriei virtuale trebuie să ia o decizie pentru a elibera un cadru din memorie pentru a copia paginia de pe disc în memorie.

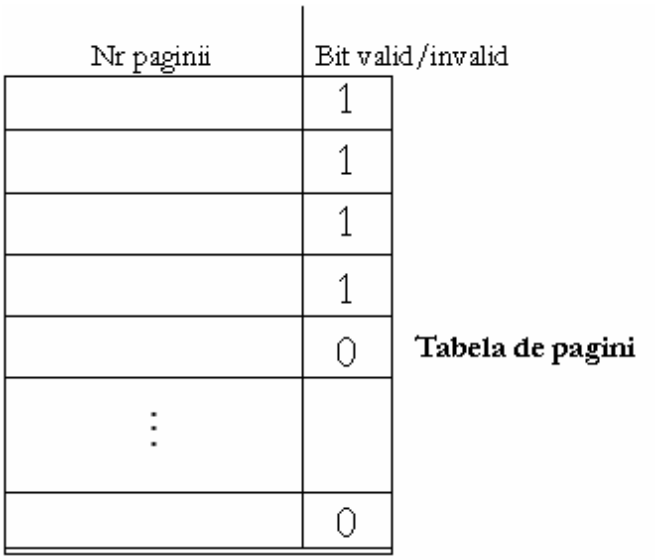



Fig. 4.16 Bitul valid/invalid din tabela de pagini

### *4.5.3 Tratarea fenomenului de page-fault*

Să analizăm în continuare mai detaliu mecanismul de funcţionare a memoriei virtuale în cazul apariţiei unui *page-fault*. În primul rând „regula" spune că cererile de pagini din memoria externă *(page-fault)* nu se produc decât rareori. Tabela de pagini necesită prezenţa unui bit „rezident" care ne arată dacă pagina respectivă este sau nu în memorie. Uneori se utilizează termenul „valid" pentru a indica prezenţa paginii în memorie (figura 4.16). O pagină „invalidă" este astfel o pagină nerezidentă sau care are o adresă ilegală. Cu toate acestea, mai logic este să avem doi biţi în tabela de pagini: unul care să indice faptul că pagina este validă iar al doilea să arate dacă pagina este rezidentă sau nu în memorie (figura 4.17).

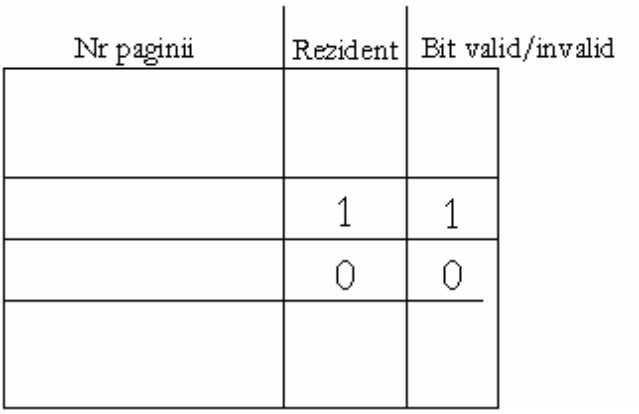



Fig. 4.17 Bitul rezident şi bitul valid din tabela de pagini

Totdeauna când apare un fenomen *page-fault*, se pune problema administrării (rezolvării) acestuia. În general, paşii care se urmează sunt (figura 4.18):

- Procesul necesită o pagină ce nu este rezidentă în memorie;
- Se verifică în tabela de pagini dacă referința de memorie este validă sau nu;
- Dacă referința este validă dar pagina nu este rezidentă, se încearcă obținerea acesteia din memoria secundară;
- Se caută şi se alocă un cadru liber (o pagină de memorie fizică neutilizată până în prezent; uneori este necesară eliberarea unei pagini de memorie);
- Se planifică o operație de acces la disc pentru a se citi acea pagină din memoria secundară în cadrul nou alocat;
- După scrierea paginii în memorie se modifică tabela de pagini; pagina este acum rezidentă în memorie;
- Se reporneşte instrucțiunea ce a generat *page-fault.*

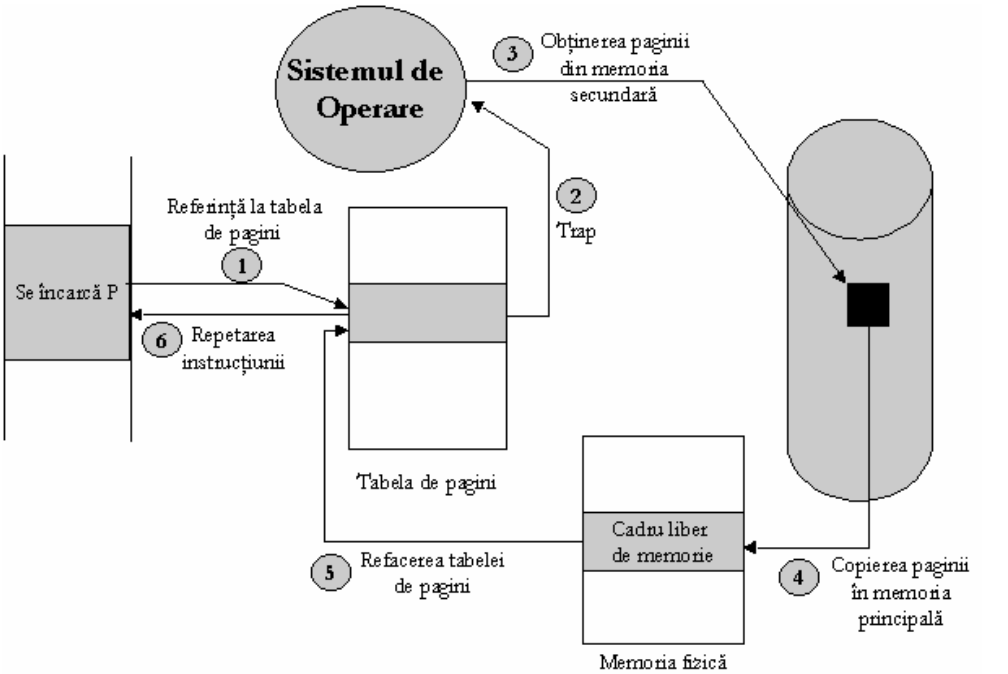



Fig. 4.18 Paşii administrării unui *page-fault*

Observăm în procesul de administrare a unui *page-fault* prezentat mai sus că în cazul în care nu există nici un cadru liber (memorie *supra-alocată*) trebuie să renunţăm la ceva deja existent în memorie. Supra-alocarea apare atunci când programele au nevoie de mai multe pagini de memorie decât cele existente fizic. Maniera de abordare şi rezolvare a problemei cercetează dacă nici o pagină fizică nu este liberă, apoi se caută una care nu este utilizată la momentul respectiv şi se eliberează, urmărind următorii paşi (figura 4.19):

1. Se caută pagina pe disc;
2. Se caută un cadru liber
    a) Dacă există, se foloseşte;
    b) Altfel, se selectează o pagină „victimă”;
    c) Se scrie pagina „victimă” pe disc;
3. Se citeşte noua pagină în cadrul eliberat. Se modifică tabela de pagini;

4. Se reporneşte procesul.

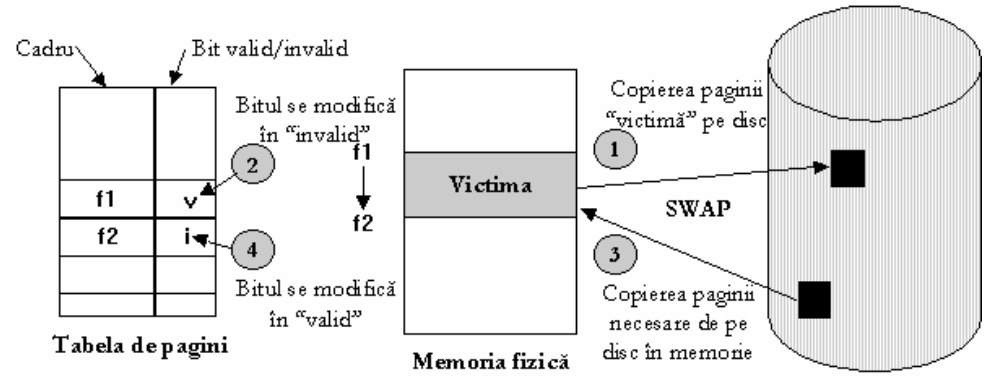



Fig. 4.19 Funcţionarea mecanismului de înlocuire a paginilor

### *4.5.4 Algoritmi de înlocuire a paginilor*

Pentru ca mecanismul de înlocuire a paginilor să funcţioneze cât mai eficient, s-a încercat să utilizeze diverşi algoritmi pentru acest proces, astfel încât să se minimizeze numărul de *page-faults*. Prezentăm în continuare trei algoritmi cunoscuţi de înlocuire a paginilor: algoritmul FIFO, algoritmul de înlocuire optimă şi algoritmul „cea mai puţin recent folosită pagină".

#### *4.5.4.1 Algoritmul FIFO*

Acest algoritm este foarte uşor de implementat din punct de vedere conceptual; el se bazează pe mecanismul cozii, (FIFO - First In / First Out) adică primul intrat este primul ieşit. Astfel, prima pagină intrată în memorie va fi şi prima eliberată din memorie. Considerând secvenţa următoare de cerere de pagini:

1,2,3,4,1,2,5,1,2,3,4,5

conform algoritmului FIFO (figura 4.20) vor apărea 10 page-faults.

Mecanismul funcţionării memoriei virtuale presupune, desigur, şi o schemă de protecţie ce restricţionează accesul proceselor (programelor) la blocuri de memorie ce aparţin altor procese (programe). Majoritatea formelor de memorie virtuală reduc, de asemenea, timpul de pornire a unui program, deoarece nu tot codul sau toate datele trebuie să fie deja în memoria fizică înainte ca programul să înceapă.

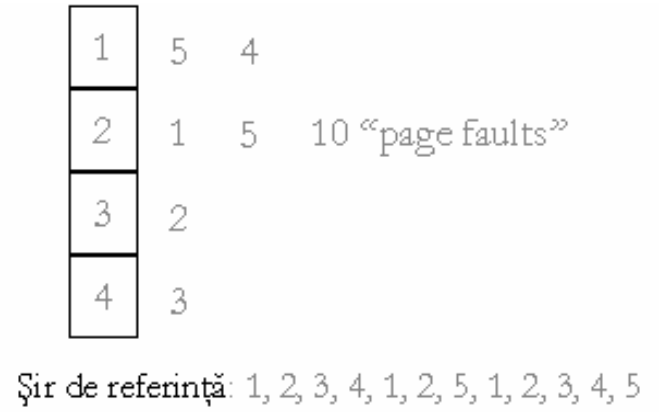



Fig. 4.20 Algoritmul FIFO de înlocuire a paginilor

### *4.5.4.2 Algoritmul de înlocuire optimă*

Acest algoritm reprezintă o politică de înlocuire a paginilor care au cea mai mică rată de *page-fault* şi constă în înlocuirea paginii care nu va fi utilizată cea mai lungă perioadă de timp. Cu toate că algoritmul ar face o înlocuire *optimă*, este imposibil de realizat în practică. Pe baza aceluiaşi exemplu de şir de pagini, în acest caz se obţin doar 6 *page-faults* (figura 4.21).

### *4.5.4.3 Algoritmul LRU (Least Recently Used)*

Algoritmul LRU, după cum îi spune şi numele, înlocuieşte pagina cel mai puţin recent folosită. Cu alte cuvinte, se înlocuieşte pagina care nu a fost utilizată pentru cea mai mare perioadă de timp. În general s-a observat că rezultatele sunt bune, dificultatea apare însă la implementare.

Există două variante mai cunoscute de implementare: una care aplică un aşa numit *time-stamp* pentru pagini, care înregistrează ultima utilizare şi alta care utilizează o stivă de pagini, în care se scot paginile utilizate şi se introduc în vârful stivei. Ambele metode necesită facilităţi hardware (trebuie făcută reactualizare după fiecare instrucţiune), de aceea este rar utilizată în practică. Considerând cazul aceluiaşi exemplu, conform algoritmului LRU se obţin 8 *page-faults* (figura 4.22).

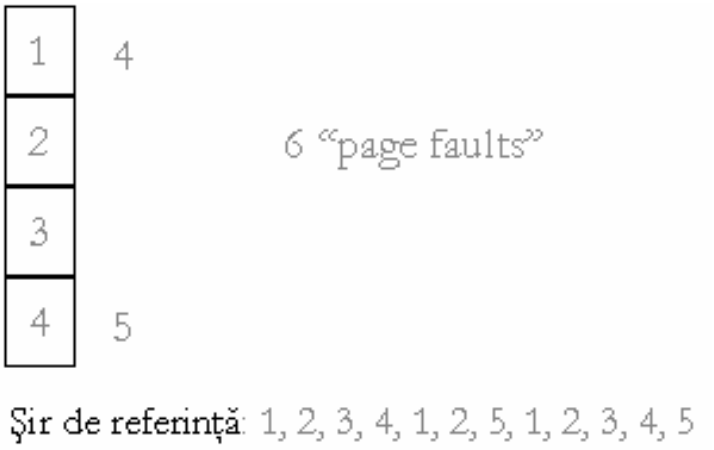



Fig. 4.21 Algoritmul de înlocuire optimă a paginilor

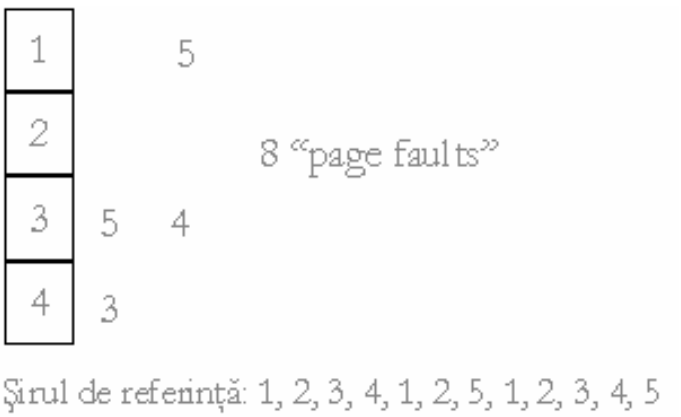



Fig. 4.22 Algoritmul „LRU” de înlocuire a paginilor

### *4.5.5 Suprapuneri*

Nu însă partajarea între procese sau între programe este adevăratul motiv pentru care a fost inventat mecanismul memoriei virtuale. Dacă un program devine prea mare pentru memoria fizică, este sarcina programatorului să îl facă să încapă în ea (caz în care apar aşa-numitele suprapuneri - *overlays*). Ideea acestor suprapuneri a apărut demult, în timpul celei de-a doua generaţii de calculatoare, atunci când disponibilul de memorie era foarte limitat. Problema era atunci cum să faci să încapă un

program de 32 KB într-un spaţiu de memorie de 16 KB. Cu toate că astfel de probleme nu mai apar, suprapunerile sunt încă utilizate. Ideea de bază este aceea de a împărţi programul în mai multe module logice independente. Să considerăm exemplul unui program cu 6 module de câte 64 KB. Modulul 1 conţine logica de control principală şi datele comune întregului program. Modulul 2 validează datele de intrare. Modulul 3 este apelat în cazurile de eroare de modulul 2. Modulul 4 generează unele calcule intermediare. Modulul 5 genereaza calculele pe anumite cazuri speciale, iar modulul 6 afiseaza rezultatele, deci este nevoie de el doar în final. În mod evident, pentru ca programul să funcţioneze, modulul 1 trebuie să rămână rezident permanent în memorie.

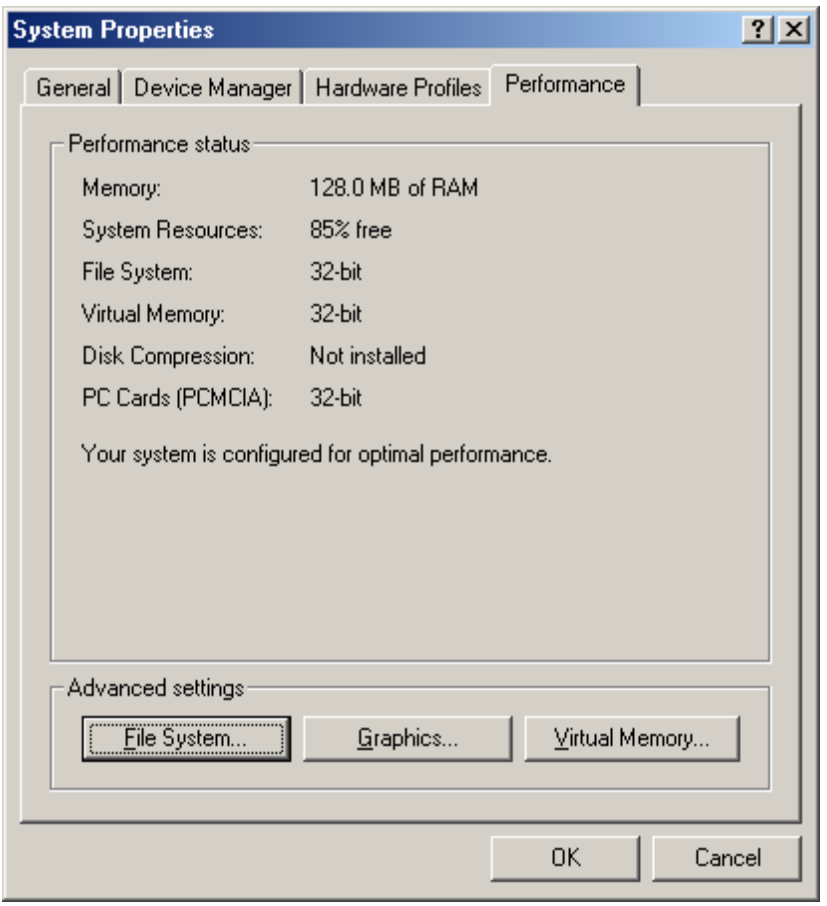



Fig. 4.23 Fereastra *System Properties – Performance* pentru Windows Me/2000

Dacă nu apar erori la intrare, nu este nevoie de modulul 3. Dacă apar erori, vom avea nevoie de modulul 3, dar nu mai avem nevoie de modulele 4, 5 şi 6. În acest mod, atunci când programul îşi începe execuția, în memorie se vor afla doar modulele 1şi 2. Dacă apare vreo eroare de intrare, modulul 3 se suprapune peste modulul 2, rămânând în memorie până când este citit un nou set de date, când este suprapus peste el în memorie modulul 2. Dacă sunt introduse date corecte şi nu apar erori, modulul 4 este suprapus peste modulul 2. În continuare, dacă apare un caz special, este tratat de modulul 5 care va fi suprapus peste modulul 4. În final, pentru afişarea rezultatelor se va încărca în memorie modulul 6, suprapus peste modulul 4 sau 5 (în funcție de cazul în care ne aflăm).

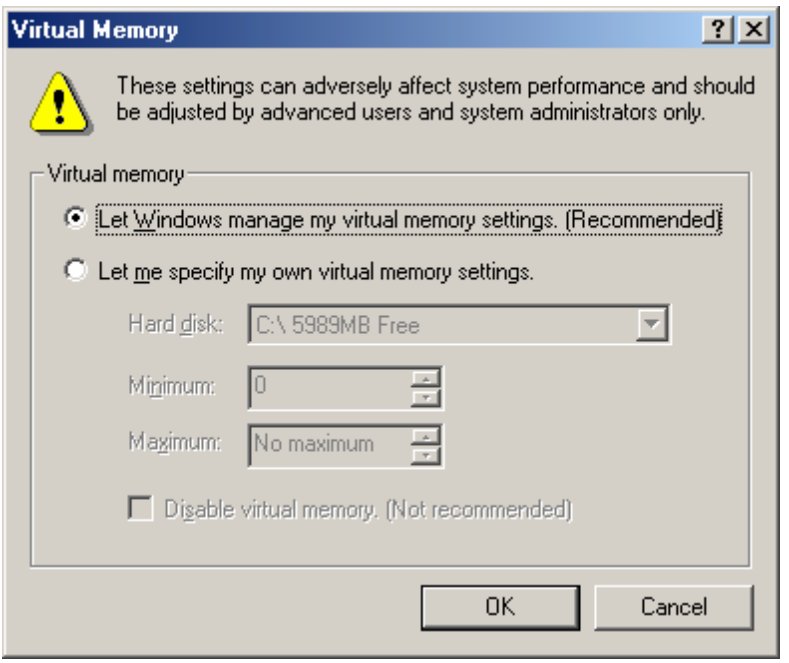



Fig. 4.24 Fereastra de setare a memoriei virtuale în Windows Me/2000

### *4.5.6 Setarea memoriei virtuale*

Memoria virtuală poate fi setată de către utilizator în mod diferit, în funcție de sistemul de operare utilizat. De exemplu, în Windows stabilirea parametrilor memoriei virtuale se face din *System Properties* → *Performance* → *Virtual Memory* (figura 4.23). Există posibilitatea alegerii

administrării automate a dimensiunii memoriei virtuale de către sistem (opțiunea *<Let Windows manage my virtual memory settings (Recommended)>*) (figura 4.24) sau specificarea explicită a dimensiunii spațiului de memorie virtuală (opțiunea *<Let me specify my own virtual memory settings>*).

În acest din urmă caz se poate specifica spațiul de pe hard-disk pe care dorim să îl rezervăm pentru memoria virtuală (spre exemplu, dacă dorim să avem un spațiu de 256 MB de memorie, vom introduce această valoare în subopțiunea *Hard-disk:*.

**Observație:** dacă se alege această setare, pentru performanțe mai bune este indicat ca dimensiunile *Minim:* şi *Maxim:* să fie setate tot la aceeaşi valoare – în cazul nostru 256 MB).

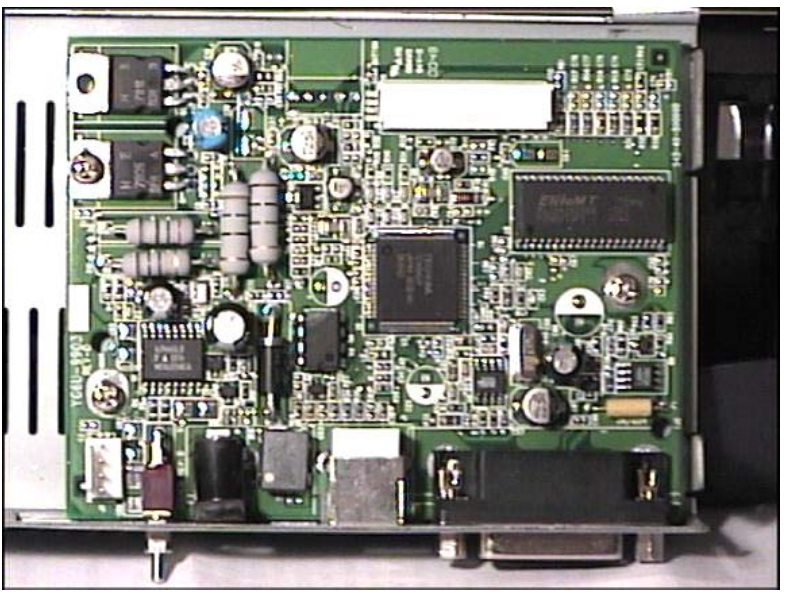


Fig. 4.25 Placa de bază a unui calculator

## 4.6 Magistrale de sistem

Arhitecturile pe baza cărora sunt construite calculatoarele nu sunt identice pentru toate sistemele de calcul. Proiectanții de calculatoare utilizează termenul de *arhitectură* pentru a descrie modalitatea de interacțiune între componentele calculatorului. Una din componentele de

bază ale unei arhitecturi o reprezintă mecanismul prin care se stabilesc legăturile dintre componentele interne ale calculatorului. De regulă, mecanismul prin care se realizează transferurile de date (fie ele instrucţiuni sau programe), semnalele de control, întreruperi, etc. este cunoscut generic sub numele de *magistrală (bus)*. Din punct de vedere arhitectural există două tipuri de magistrale: arhitectura monomagistrală *(single-bus architecture)* şi arhitectura multimagistrală *(multiple-bus architecture)*.

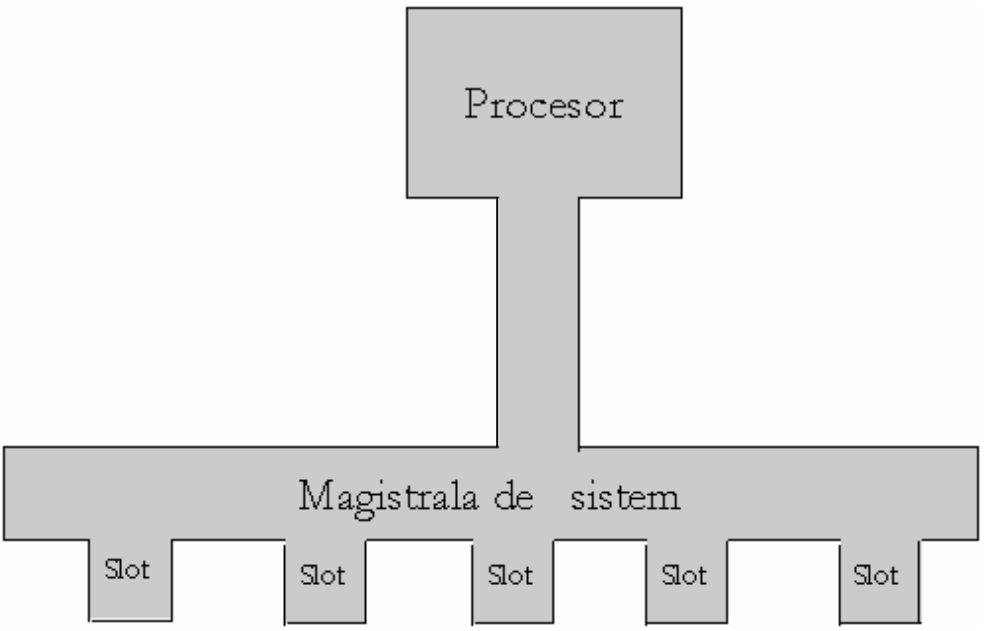



Fig. 4.26 O reprezentare schematică a magistralei şi a sloturilor unde pot fi adăugate diferite componente

### *4.6.1 Arhitectura monomagistrală*

Majoritatea calculatoarelor sunt construite pe suportul unei plăci de bază (figura 4.25), un cadru metalic ce conţine o serie de locaşe (*sloturi*) aflate în legătură, prin intermediul magistralei, cu procesorul (figura 4.26). Cipurile de memorie pot fi adăugate plăcii de bază prin introducerea acestora în locaţii speciale. De asemenea, pe placa de bază mai pot fi inserate diverse plăci adiţionale pentru conectarea dispozitivelor periferice de intrare/ieşire (spre exemplu, plăci de sunet, plăci de captură video, plăci de modem etc.) sau echipamente de stocare externă (hard-diskuri, floppy diskuri etc.). Deoarece toate aceste componente partajează o singură

magistrală comună, acest model se numeşte *arhitectură monomagistrală*. Toate tipurile de comunicaţii ce apar între componente au loc pe această magistrală (figura 4.27).

***Interfeţe de conexiune.*** Semnalele electrice de control pentru dispozitivele periferice sunt diferite între ele (diferă între imprimantă şi scanner, sau modem, etc.), de aceea fiecare dispozitiv periferic are propria *interfaţă de conexiune*. O componentă a interfeţei interacţionează cu calculatorul folosind diferite coduri interne. Cealaltă componentă comunică cu echipamentul respectiv, pe baza unui limbaj propriu. Funcţia de bază a interfeţei este aceea de asigura translatarea mesajelor între calculator şi dispozitivul periferic.

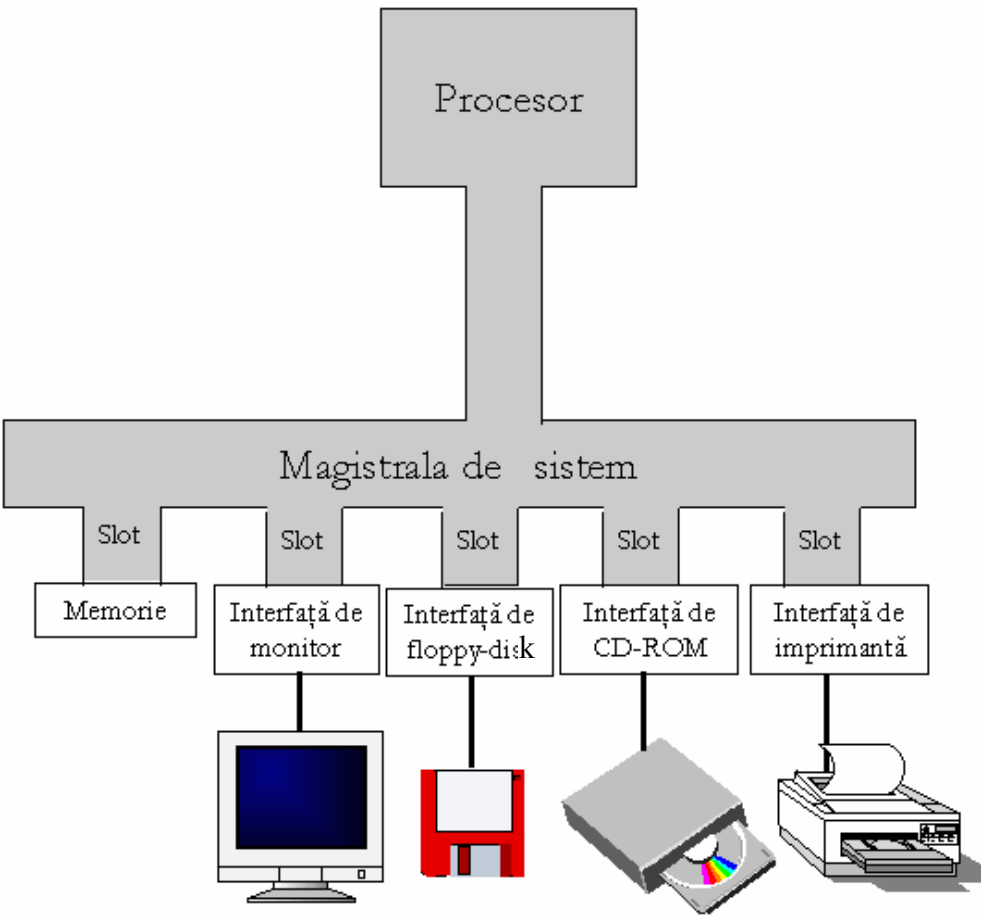



Fig. 4.27 Magistrala de sistem şi diferite interfeţe

***Canale şi unităţi de control.*** Calculatoarele personale (*micro-computerele*, cum se mai numesc) sunt destinate în general lucrului unui singur utilizator, de aceea arhitectura mono-magistrală este suficientă pentru ca operaţiunile pe calculator să decurgă în condiţii bune. Un calculator foarte puternic, de tipul *mainframe*, ar fi mult prea puternic şi mult prea costisitor pentru a fi dedicat exploatării de către un singur utilizator. Calculatoarele de tip *mainframe* oferă, de regulă, suport pentru mai mulţi utilizatori în mod concurent. Un ciclu de maşină pentru un calculator mainframe este identic cu acela al unui micro-calculator în sensul în care procesorul desfăşoară aceeaşi activitate de bază: aduce instrucţiunile din memorie şi le execută pe rând. Un element cheie în administrarea mai multor utilizatori este acela de a elibera procesorul de responsabilitatea controlului dispozitivelor de intrare/ieşire. Controlul permanent al dispozitivelor de intrare/ieşire implică o serie de funcţii logice de selectare a drumului pe care să "curgă" datele, de numărare a caracterelor sau de calcul a adreselor de memorie. Deoarece procesorul este singura sursă de control a acestor operaţii pe un sistem microcomputer, procesorul trebuie să se implice în mod direct asupra fiecărei operaţii de intrare/ieşire. În timp ce controlează intrarea/ieşirea, procesorul nu este disponibil pentru alte operaţii, cum ar fi execuţia instrucţiunilor.

În cazul computerelor mainframe, acestea predau responsabilitatea controlului operaţiilor de intrare/ieşire aşa numitelor *canale*. Un *canal* reprezintă un micro (mini)-computer cu procesor propriu ce poate asigura funcţiile logice de control în paralel cu procesorul principal al calculatorului. Această abordare face ca procesorul principal să se elibereze de operaţiile de control ale dispozitivelor periferice, deci îşi poate concentra întreaga atenţie asupra celorlalte activităţi. Unele dintre funcţiile de intrare/ieşire sunt dependente de dispozitive; de exemplu, controlul unui braţ de acces la disc este o problemă de hard-disk, în timp ce o imprimantă ink-jet trebuie să convertească reprezentarea unui caracter conform unei machete de puncte de cerneală. Alte funcţii (selecţia unei căi pentru date, calcularea adreselor de memorie etc.) sunt însă comune tuturor dispozitivelor periferice (independente de acestea). Soluţia este ca funcţiilor dependente de dispozitive să le fie asociate câte o unitate de control I/E. Fiecare dispozitiv fizic va avea astfel o astfel de unitate de control I/E.

### *4.6.2 Arhitectura multimagistrală*

Arhitectura monomagistrală are o serie de probleme atunci când este vorba despre un sistem multiutilizator. Comunicarea pe canale implică spre exemplu deplasarea datelor între memorie şi un dispozitiv periferic. Procesorul calculatorului administrează datele în memorie. Accesul simultan la memorie pentru procesor şi pentru un canal nu este posibil pe un sistem mono-magistrală deoarece în acest caz se oferă o singură cale fizică de comunicaţie. Accesul simultan implică existenţa unor căi de comunicaţie distincte, astfel încât majoritatea computerelor mainframe folosesc arhitectura *multimagistrală*. Să luăm cazul unui canal; în mod normal acesta este unit de procesor şi de memorie prin intermediul a două magistrale. Odată cu începerea unei operaţii de intrare/ieşire, procesorul principal trimite o comandă *start I/E* de-a lungul magistralei de comenzi către procesorul canalului.

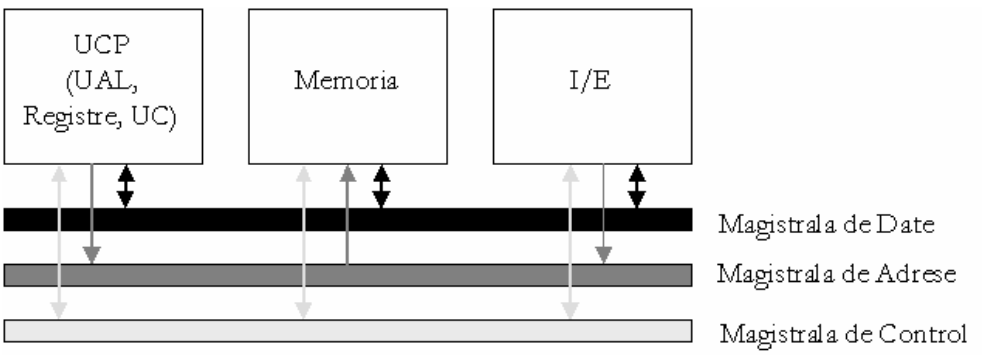



Fig. 4.28 Magistrala de sistem este formată din magistrala de date, magistrala de adrese şi magistrala de control

Drept răspuns, canalul îşi asumă responsabilitatea pentru continuarea operaţiei de intrare/ieşire, stabilind o legătură cu echipamentul periferic şi controlând transferurile de date în memorie de-a lungul magistralei de date. În acest timp, procesorul principal poate prelua alte activităţi.

Un *canal* reprezintă un dispozitiv asincron independent, cu procesor propriu, ce controlează operaţiile de intrare/ieşire. Deoarece canalul şi calculatorul sunt dispozitive independente, procesorul principal nu are cum să ştie dacă operaţia de intrare/ieşire s-a terminat în cazul în care canalul nu îl înştiinţează de acest lucru. În astfel de cazuri, în timp ce ultimul caracter din şirul de date a plecat pe drumul său pe magistrala de date, procesorul canalului trimite procesorului principal un semnal, denumit *întrerupere*. În

momentul recepționării unei întreruperi, procesorul ştie că operația de intrare/ieşire respectivă s-a terminat şi va acționa în consecință.

Magistrala de sistem se împarte în mai multe submagistrale, fiecare dintre acestea fiind utilizată pentru transportul anumitor şiruri de biți. În figura 4.28 putem observa trei tipuri diferite de magistrale, magistrala de date, magistrala de adrese şi magistrala de control, toate acestea alcătuind *magistrala de sistem*.

## 4.7 Memoria externă

Informațiile procesate cu ajutorul calculatorului trebuie să fie stocate cu ajutorul anumitor dispozitive de memorare. Acest tip de memorie se numeşte *memorie externă* şi vom prezenta în continuare câteva caracteristici ale hard-discurilor, discurilor optice şi unităților de dischetă.

### *4.7.1 Hard-diskul*

Discul dur, sau hard-diskul a fost introdus pentru prima dată de către firma IBM sub denumirea de disc *Winchester* (3340), în scopul de a îmbunătăți unitățile *DASD (Direct Access Storage Device)* de stocare existente până atunci. Toate datele procesate de (şi în) calculator pot fi înmagazinate pe hard-diskuri pentru a fi folosite mai târziu. Atunci când calculatorul beneficiază de o oprire corectă, informația este stocată pe hard disk, fiind disponibilă din nou când repornim calculatorul. Din punct de vedere fizic, un hard disk este format din unul sau mai multe platane. Aceste platane sunt împărțite în cilindri, piste şi sectoare (figura 4.29). Definițiile acestor denumiri sunt prezentate în tabelul următor:

| | |
|---|---|
| ***Platane*** | Sunt, de regulă, discuri dintr-un "oxid" sau "film subțire" învelite cu o folie de aluminiu, mediu pe care sunt stocate efectiv datele. |
| ***Piste*** | Cercuri concentrice existente pe fiecare platan unde sunt stocate magnetic datele. |

| | |
|---|---|
| ***Capete de citire/scriere*** | Există, de regulă, o pereche de capete de citire/scriere pe fiecare platan (unul deasupra şi unul dedesubt) iar capetele de citire/scriere se deplasează împreună. |
| ***Sector*** | Fiecare pistă este împărțită în unități de dimensiune mai mică denumite sectoare.Un sector are, de regulă, 512 octeți. La un hard-disk există un număr care defineşte numărul de sectoare de pe o pistă (32, 64 etc). |
| ***Ax de rotație*** | Este axul ce trece prin mijlocul fiecărui platan. Platanele sunt rotite în jurul axului în mod asemănător învârtirii unei roți în jurul axului. |
| ***Cilindru*** | Mulțimea tuturor pistelor de pe toate platanele care sunt la aceeaşi distanță de axul de rotație. |

Capetele de citire/scriere şi brațele de susținere sunt sigilate împreună cu platanul pentru a fi protejate la praf sau diverse alte particule ce pot influența ulterior mecanismul extrem de fin de citire/scriere. Dimensiunea stratului de oxid de pe suprafața platanului este de aproximativ 2,5 μm iar capul de citire/scriere se deplasează deasupra sau dedesubtul platanului la o distanță infimă, de numai 0,5 μm (drept comparație, un fir de păr uman are o grosime de aproximativ 50 μm).

Dimensiunile actualelor hard-diskuri ajung la capacități de ordinul zecilor de GB, dar dacă ne uităm în trecut, primul PC era echipat doar cu discuri floppy iar popularitatea de mai târziu a calculatorului PC-AT s-a datorat parțial şi existenței hard-diskului ST506 cu care calculatorul era echipat standard. În continuare, unitățile de disc IDE (Intelligent Drive Electronics) de la firma Compaq au ajuns la capacități mult mai mari şi cu performanțe net superioare. Între timp şi interfața *SCSI* (*Small Computer System Interface*, pronunțat ca în englezescul „scuzzy") a devenit un standard unanim acceptat de către producătorii şi utilizatorii de calculatoare. Această interfață a fost adoptată şi pentru alte dispozitive decât hard-diskurile magnetice, dar prețul acesteia este mult mai ridicat decât al interfeței IDE, de aceea nu vom întâlni interfețe SCSI de regulă, decât la

calculatoare de tipul serverelor, unde este nevoie de performanțe superioare şi unde costurile nu sunt aşa importante.

Unele hard-diskuri au 17 sectoare pentru fiecare pistă şi 980 de piste pe o suprafață. În aceste condiții,capacitatea maximă de stocare (C) a unui hard-disc se poate calcula astfel:

**C = Npl * Np * Ns * Ds,** unde:

Npl = numărul de platane ale hard-diskului;

Np = numărul de piste de pe un platan;

Ns = numărul de sectoare de pe o pistă;

Ds = dimensiunea unui sector.

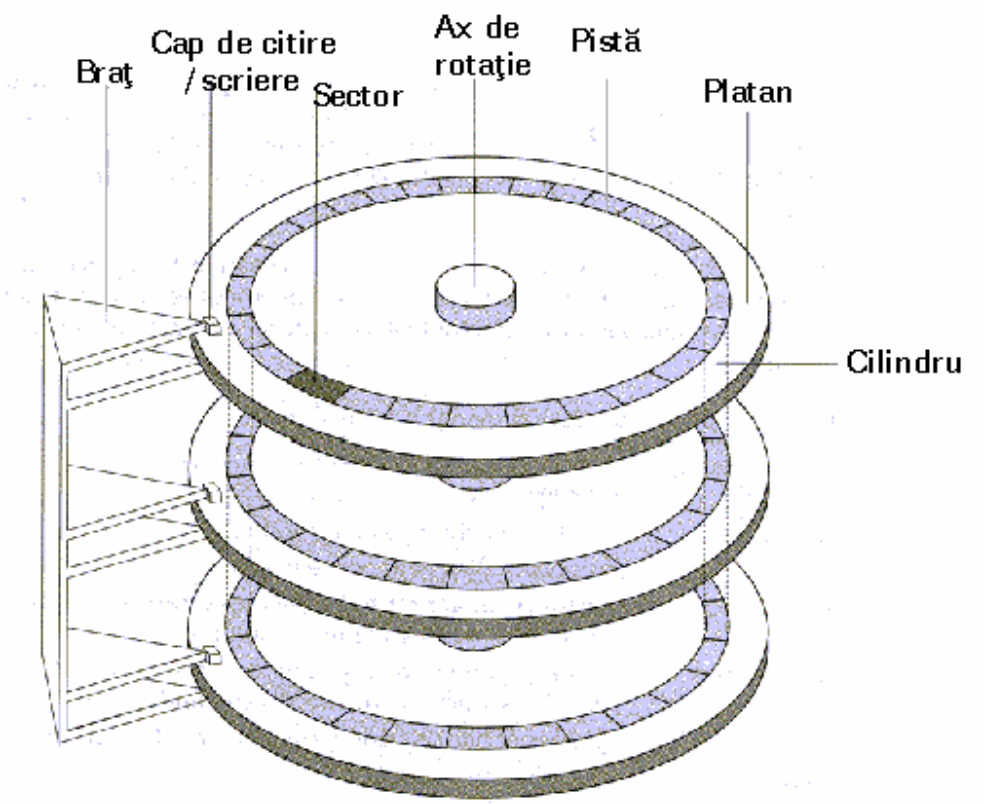



Fig. 4.29 Imagine detaliată a unui hard-disk

Accesul la informaţia de pe un hard-disc se realizează prin specificarea controller-ului de hard-disk a numărului de cilindru, sectorul şi capul de citire/scriere ce trebuie accesat.

### *4.7.2 Discuri optice*

Modalitatea de stocare folosind dicurile optice îşi are începuturile o dată cu înlocuirea discurilor de vinilin cu compact discurile din industria muzicală. În momentul de faţă există mai multe tipuri şi formate de compact discuri, pe care le prezentăm în continuare.

#### *4.7.2.1 CD-ROM (Compact Disk, Read Only Memory)*

Reprezintă un tip foarte popular de mediu de stocare movibil ce a fost conceput iniţial pentru înmagazinare de date audio dar care s-a extins rapid în lumea calculatoarelor personale ca suport pentru stocarea datelor. Succesul CD-ROM-ului poate fi atribuit capacităţii de stocare, durităţii şi, nu în ultimul rând, preţului redus. Datorită răspândirii acestui format de stocare, unităţile de CD-ROM reprezintă echipamente standard prezente pe majoritatea calculatorelor personale.

CD-urile au un diametru de 120 mm, 1,2 mm grosime şi pot stoca până la 800 MB de informaţie. Ele sunt construite dintr-un strat de plastic, un strat de metal reflectiv şi un înveliş de lac. CD-urile reprezintă un mediu optic de stocare diferit de mediile magnetice de genul floppy disk, hard disk sau discuri Zip. O unitate CD-ROM este alcătuită dintr-un motor, ansamblu laser, mecanism de rotaţie şi circuite electronice. Motorul unităţii învârte motorul CD-ROM-ului la viteza necesară astfel încât ansamblul laser să poată citi informaţia. Ansamblul laser constă dintr-un laser şi lentile optice; acest ansamblu citeşte CD-ROM-ul în timp ce acesta se roteşte. Mecanismul de rotaţie este un motor ce deplasează lentilele în poziţia corectă pentru a accesa o zonă anume a CD-ROM-ului. Circuitele electronice asigură transferul informaţiei citite de pe CD-ROM către calculator prin intermediul unei magistrale.

Unităţile CD-ROM sunt clasificate în funcţie de tipul carcasei, tipul magistralei şi viteza de citire. Unităţile CD-ROM pot fi montate în interiorul unui computer sau extern. Unităţile interne preiau puterea de alimentare de

la sursa de putere a calculatorului şi sunt conectate la magistrala de date a calculatorului cu ajutorul unui cablu.

Unităţile externe au carcasa lor proprie de protecţie şi propria sursă de putere. Se conectează la calculator direct la un port extern al computerului (USB, FireWire, sau paralel) sau la un controller instalat într-unul dintre slot-urile de extensie ale calculatorului (de regulă SCSI). O magistrală este o conductă de comunicaţie între un calculator şi perifericele instalate. Conexiunile clasice interne sunt IDE şi SCSI. Majoritatea plăcilor de bază au controllere IDE instalate şi vor recunoaşte automat o unitate CD-ROM instalată. Cablurile IDE de comunicaţie au 40 de pini ce conectează unitatea de placa de bază. Unele plăci oferă soluţii înglobate SCSI, pe când majoritatea necesită un adaptor SCSI cumpărat şi instalat separat.

### *Viteze de citire*

Viteza de citire a unei unităţi CD-ROM determină rata de transfer a informaţiei de la CD-ROM la calculator. În principiu, cu cât viteza de citire este mai mare, cu atât transferul datelor este mai rapid. Viteza de citire a unei unităţi CD-ROM este multiplu de 150 kb şi este succedată de un "x". O unitate CD-ROM cotată cu viteza de citire 1x înseamnă că citeşte datele cu o viteză de 150 kb pe secundă. Astfel, un CD-ROM 10x citeşte 1500kb pe secundă. Fluxurile de date de pe un CD (video) pot necesita viteze de minim 12x. De asemenea, rata de citire a unui CD-ROM nu este totdeauna uniformă. O unitate CD-ROM 100x poate ajunge la această viteză în apropiere de centrul CD-ROM şi doar la viteza 40x pe pistele exterioare.

### *Inscripţionarea CD-urilor*

În momentul de faţă există două tipuri de CD recordere:

♦ ***CD-R*** *(Compact Disk Recordable)* reprezintă prima tehnologie apărută. CD-urile comerciale produse în cantităţi mari sunt realizate prin presare. Această tehnică produce adâncituri pe CD ce sunt citite de razele laser. Tehnologia CD-R foloseşte altă strategie pentru scrierea informaţiei pe CD. Mediul CD-R adaugă un strat de colorant între stratul de aluminiu şi cel de plastic. Acest strat de colorant este translucent şi permite luminii să ajungă la stratul de aluminiu de unde se relectă înapoi. Atunci când o unitate CD-ROM scrie informaţii pe un mediu CD-R se foloseşte un laser pentru a arde zone în stratul colorant pentru a crea pete opace, care nu reflectă

lumina. Atunci când CD-ul este citit, ansamblul laser recepţionează reflecţiile doar din zonele translucente. Această suprafaţă transformă reflecţia/nonreflecţia în biţi de date. Sunt folosite două lasere: unul de citire şi unul de scriere. O unitate CD-R cu caracteristicile 24x/40x are o viteză de scriere de 24x şi una de citire de 40x.

♦ ***CD-RW*** *(Compact Disk ReWritable).* Ca şi unităţile CD-R, unităţile CD-RW necesită utilizarea unui tip special de disc. Mediul CD-RW este diferit în sensul că include un strat modificator de stare între starturile de plastic şi cel de aluminiu. Acesta este compus din anumite elemente chimice care îşi modifică starea fizică la anumite temperaturi. Când raza laser de scriere este folosită pentru a scrie informaţii pe disc, zone ale amestecului sunt topite prin supraîncălzire şi menţinute în această stare printr-o răcire rapidă. Aceste zone topite sunt opace şi nonreflective. La unităţile CD-RW apare cea de-a treia rază laser: aceea de ştergere. Raza de ştergere este utilizată pentru a aduce înapoi în stare cristalină amestecul. O unitate CD-RW cu caracteristicile 24x/12x/40x are o viteză de scriere de 24x, de rescriere de 12x şi de citire de 40x.

Disponibilitatea şi costul scăzut al inscriptoarelor de CD şi al mediilor de scriere a făcut ca mulţi utilizatori să poată să-şi facă propriile compilaţii muzicale, astfel încât, prima dată de la apariţia CD-urilor audio utilizatorii au putut să-şi creeze propriile CD-uri cu muzică la o calitate de înregistrare digitală. Extragerea audio digitală (EAD) face acest lucru posibil. EAD reprezintă procedeul prin care se copiază un CD audio pe alt mediu de stocare (variantă de CD-R, CD-RW, etc.), menţinând calitatea audio în forma digitală originală. Această tehnică mai este cunoscută şi sub numele de ripping, existând o serie de aplicaţii ce realizează acest lucru (o performantă aplicaţie de tip open-source fiind Cdex (figura 4.30), http://cdexos.sourceforge.net ), copiind melodii de pe un CD pe hard-diskul calculatorului şi apoi pe un CD inscriptibil, putând astfel păstra melodiile în format digital.

În momentul în care un CD player citeşte un CD în format audio (RebBook CD), unitatea de citire trebuie să realizeze o conversie digital-analog în timp real astfel încât sunetul să poată fi auzit prin difuzoare. Înainte de extragerea audio digitală, sunetul trebuie să fie re-înregistrat şi redigitizat în momentul în care utilizatorul doreşte să înregistreze melodia pe un CD inscriptibil. Această conversie digital-analog-digital este consumatoare de resurse şi pot apărea, eventual, şi unele zgomote de fond. În funcţie de calitatea unităţii de CD-ROM, a calculatorului şi a software-ului utilizat, procedeul de extragere audio

digitală poate realiza copii digitale ale melodiilor în timp scurt. Copierea unui cântec de pe un CD audio este un proces diferit de copierea unui fişier de pe un CD-ROM. O melodie este stocată într-o locaţie specială denumită *track (pistă)* iar unitatea de CD-ROM citeşte un astfel de *track* cu ajutorul mecanismului laser de care dispune. Informaţia audio este trimisă sub forma unui flux către unitate şi din această cauză trebuie să existe o aplicaţie specială pentru procedeul EAD. Software-ul respectiv citeşte informaţia brută a unui track sub forma unui flux şi colectează informaţiile într-un fişier de tip WAV. Acest fişier WAV poate fi apoi citit pe calculator şi copiat înapoi pe un CD inscriptibil împreună cu late melodii pentru a crea un CD audio personalizat. Fişierele în format WAV au dimensiuni foarte mari fiind fişiere necompresate; de regulă aceste fişiere sunt compresate în alte formate (fişierele au dimensiuni mult mai mici în acest caz) precum MP3 sau Ogg Vorbis cu păstrarea unei calităţi decente a melodiilor.

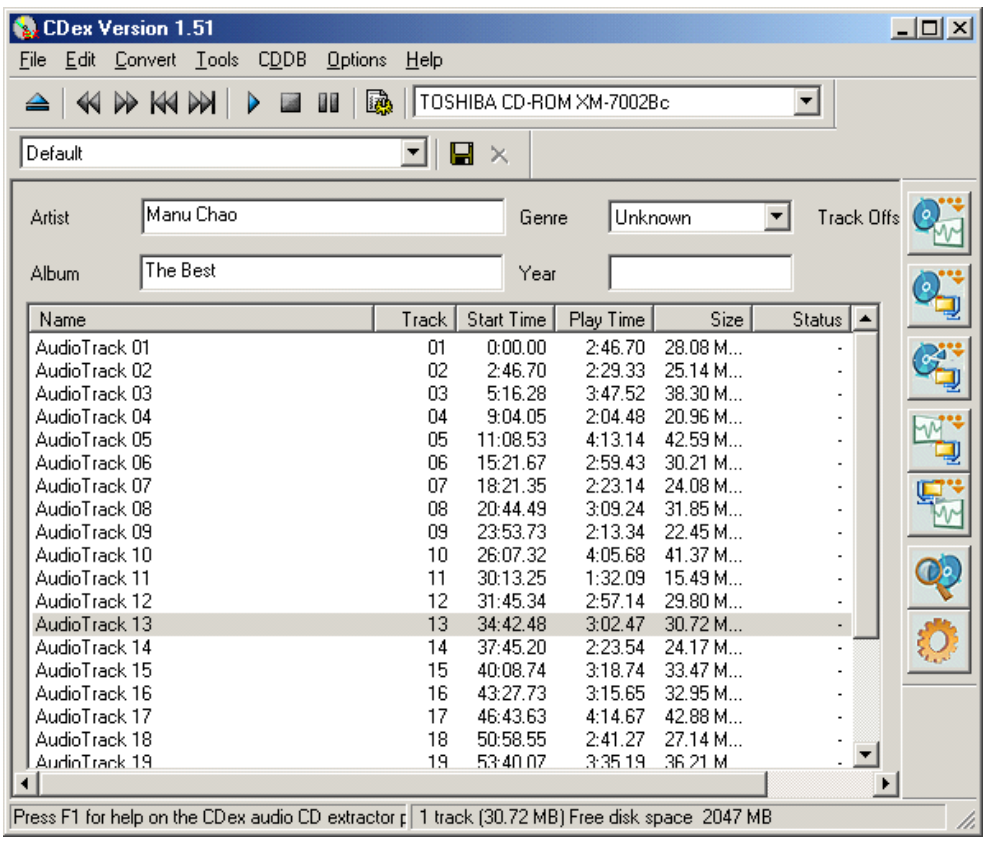

CDex Version 1.51
File Edit Convert Tools CDDB Options Help
TOSHIBA CD-ROM XM-7002Bc
Default
Artist Manu Chao
Genre Unknown
Track Offs
Album The Best
Year
Name Track Start Time Play Time Size Status
AudioTrack 01 01 0:00.00 2:46.70 28.08 M... -
AudioTrack 02 02 2:46.70 2:29.33 25.14 M... -
AudioTrack 03 03 5:16.28 3:47.52 38.30 M... -
AudioTrack 04 04 9:04.05 2:04.48 20.96 M... -
AudioTrack 05 05 11:08.53 4:13.14 42.59 M... -
AudioTrack 06 06 15:21.67 2:59.43 30.21 M... -
AudioTrack 07 07 18:21.35 2:23.14 24.08 M... -
AudioTrack 08 08 20:44.49 3:09.24 31.85 M... -
AudioTrack 09 09 23:53.73 2:13.34 22.45 M... -
AudioTrack 10 10 26:07.32 4:05.68 41.37 M... -
AudioTrack 11 11 30:13.25 1:32.09 15.49 M... -
AudioTrack 12 12 31:45.34 2:57.14 29.80 M... -
AudioTrack 13 13 34:42.48 3:02.47 30.72 M... -
AudioTrack 14 14 37:45.20 2:23.54 24.17 M... -
AudioTrack 15 15 40:08.74 3:18.74 33.47 M... -
AudioTrack 16 16 43:27.73 3:15.65 32.95 M... -
AudioTrack 17 17 46:43.63 4:14.67 42.88 M... -
AudioTrack 18 18 50:58.55 2:41.27 27.14 M... -
AudioTrack 19 19 53:40.07 3:35.19 36.21 M... -
Press F1 for help on the CDex audio CD extractor p
1 track (30.72 MB) Free disk space 2047 MB

Fig. 4.30 Fereastra aplicaţiei *Cdex*, utilitar pentru extragere audio digitală şi conversie în format MP3

Formate utilizate pentru scrierea CD-urilor

Firmele Sony şi Phillips au conceput formatul standard pentru compact discurile audio la începutul anilor 1980. În acest sens există două tipuri de standarde principale pentru CD:

♦ ***Standarde Logice***

Un standard logic defineşte modalitatea de stocare a informației pe disc. CD-urile şi alte discuri utilizează o serie de piste şi sectoare pentru a stoca date pe disc. Din punct de vedere logic, standardul reprezintă structura sistemului de fişiere. În prezent standardul ISO 9660 *(High Sierra)* este standardul unanim acceptat. CD-urile create cu acest format pot fi accesate pe majoritatea platformelor şi sistemelor de operare existente astăzi în lume.

Alte formate sunt: *Rock Ridge (UNIX)*, *HFS (Macintosh)* şi formatul hibrid *HFS/ISO*. Două standarde care îmbunătățesc standardul ISO 9660 sunt *JOLIET* şi *UDF*. *JOLIET* este varianta Microsoft pentru ISO 9660 ce extinde mărimea maximă a numărului de caractere ale unui nume de fişier de la 8 la 64. *Universal Disc Format (UDF)* reprezintă un standard mai nou special conceput pentru înmagazinarea datelor.

♦ ***Standarde fizice***

Standardele fizice definesc modul de scriere a informației pe CD (locul unde se scrie această informație). Majoritatea formatelor cad în categoria aşa numitelor *"Cărți colorate"*. Această caracteristică provine de la faptul că, atunci când Phillips şi Sony au adus formatul pentru CD-ul audio, primele caracteristici ale acestuia au fost publicate într-o carte cu o copertă roşie, initulată ***Red Book***. Standardele "colorate" sunt:

***- Red Book – Compact Disk - Digital Audio (CD-DA)*** este formatul pentru CD-uri audio. Specifică modul de aranjare a cântecelor în piste pe disc.

***- Yellow Book*** – A fost dezvoltat inițial ca format pentru date. Permite scrierea datelor ca fişiere pe disc.

***- Green Book*** – Firma Phillips a creat în 1986 acest format pentru noile compact discuri denumite *CD-I (Compact Disk Interactive)*. Acest format a fost special conceput pentru sincronizarea datelor audio şi video pentru aplicații multimedia.

***- Orange Book*** – Acest standard (1990) defineşte formatul fizic pentru discurile inscriptibile. Are trei părți:

- (1) pentru discuri Magneto-optice (MO),

- (2) pentru discuri de tip Write Once (WO) şi
- (3) pentru discuri reinscriptibile.

***- White Book*** – Reprezintă metoda de înregistrare MPEG1 audio-video pe un video CD (VCD). Aceste discuri necesită un player specializat sau o aplicație software pentru accesare datorită compresiei înalte a informației.

***- Blue Book*** – Standardul *Blue Book* este pentru formatul discurilor E-CD (Enhanced-CD).

| Caracteristica | CD | DVD |
|---|---|---|
| Diametru | 120 mm | 120 mm |
| Grosime | 1,2 mm | 0,6 mm |
| Distanta între piste | 1,6 mm | 0,74 mm |
| Lungimea minima a cavitatilor | 0,834 mm | 0,4 mm |
| Lungimea de unda a razei laser | 780 nm | 640 nm |
| Capacitate (pe strat) | 0,64 GB | 4,7 GB |
| Numar de straturi | 1 | 1,2,3 sau 4 |

Fig. 4.31 O comparație între caracteristicile discurilor CD şi DVD

### *4.7.2.2 DVD (Digital Versatile Disk – Digital Video Disk)*

*DVD (Digital Versatile Disk)* reprezintă cea mai nouă tehnologie bazată pe tehnologia de tip CD-ROM. Discurile video digitale, aşa cum se mai numesc (datorită intenției inițiale cu care au fost create, aceea de stocare a filmelor), încep treptat să preia locul şi rolul unităților CD-ROM şi CD-RW în stocarea informațiilor. Un disc DVD are aceeaşi dimensiune fizică precum CD-ROM-ul dar poate stoca o cantitate mult mai mare de informație. Capacitatea de a stoca mai multă informație provine de la faptul

că oferă mai mult spaţiu de stocare printr-o tehnică de înregistrare de mai mare densitate şi accesul la mai multe straturi fizice pe acelaşi mediu. În figura 4.31 este prezentat un tabel cu diferenţele dintre caracteristicile discurilor CD şi DVD. Ca şi în cazul CD-ROM-ului, iniţial a apărut varianta DVD ce poate fi doar citită ca modalitate de stocare pentru filme şi jocuri pentru calculator cu facilităţi grafice deosebite. Ulterior, discurile DVD-RW (reinscriptibile) au apărut disponibile pentru aplicaţii audio.

Unităţile DVD (în figura 4.32 sunt prezentate diverse caracteristici posibile ale discurilor DVD) sunt asemănătoare cu cele CD-ROM. În plus conţin decodoare. Toate player-ele DVD şi unele unităţi pentru computer au un decodor MPEG-2 pentru decompresia datelor video într-un format ce poate fi afişat pe ecranul televizorului sau pe monitor. Unele playere DVD au decodoare audio pentru Dolby Digital 5.1 sau DTS. Laserul utilizat într-un DVD player este diferit de cel folosit într-o unitate CD-ROM.

| Straturi/faţă | Nr feţe | Capacitate | Lungime(în ore) |
|---|---|---|---|
| 1 | 1 | 4.7 | 2 |
| 2 | 1 | 8.5 | 4 |
| 1 | 2 | 8.7 | 4.5 |
| 2 | 2 | 17 | 8 |

Fig. 4.32 Diverse caracteristici ale discurilor DVD

Pentru inscripţionare sunt disponibile mai multe formate DVD. Acestea sunt:

- ***DVD-R – DVD-R (recordable)*** – este similar cu tehnologia CD-R în sensul că discul poate fi scris o singură dată. Nu este practică pentru consumatori datorită preţurilor ridicate ale unităţilor şi discurilor DVD;
- ***DVD-RAM*** – Utilizează tehnologia "Random Access Memory" (RAM) ce permite scrierea şi rescrierea unui disc de aproximativ 100.000 de ori. DVD-RAM utilizează o tehnologie similară cu cea CD-RW şi stochează 4.7GB de date pe fiecare faţă a discului; apar însă probleme legate de compatibilitate.
- ***DVD-RW*** – Tehnologia DVD-RW (re-writable) este concepută pentru a rezolva problema compatibilităţii. Permite

rescrierea de aproximativ 1000 de ori. Discurile sunt compatibile cu majoritatea unităților DVD-ROM de pe piață.

- ***DVD+RW*** – Tehnologie de ultimă oră (HP+). Asemănătoare cu DVD-RW. Prețuri accesibile (sub $600) – posibil viitor standard pentru înregistrarea DVD.

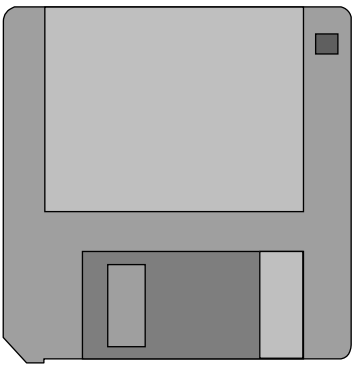


Fig. 4.33. Discheta de 3,5 inci şi capacitate 1,44 MB

### *4.7.3 Discuri floppy*

Popularele dischete, sau discurile floppy, au rămas într-un fel mult în urmă față de dezvoltările tehnice legate de hard-diskuri, discuri CD sau DVD. În timp ce alte medii de stocare au capacități foarte mari, de sute de MB sau zeci de GB, populara dischetă de 3,5 inci a rămas la capacitatea de 1,44 MB (figura 4.33) de ani buni. Nu a fost însă întotdeauna aşa; primele unități de disc flexibil introduse de IBM erau de 8 inci dimensiune, având o capacitate de stocare de 128 KB, fiind utilizate ca dispozitiv de memorie secundară pentru a încărca microcodul în cazul micro-calculatoarelor. Imediat după introducerea primului IBM-PC, firmele americane IMSAI şi Altair au început să producă computere destinate uzului casnic ce incorporau procesoare Intel 8080, magistrale de comunicație S100 şi enități de disc flexibil.

Un alt pas important în dezvoltarea unităților de discuri floppy a avut loc în momentul înlocuirii dischetelor de 5,25 inci şi 8 inci care erau mai mari, cu capacități mai reduse şi mai puțin fiabile. Dischetele de 3,5 inci au 80 de piste cu câte 9 sectoare de câte 1 KB, rezultând o capacitate de stocare

de 1,44 MB. Spre deosebire de hard-diskuri, capul de citire este în contact direct cu suprafața magnetică a discului tot timpul, ce creează uneori probleme, chiar dacă viteza de rotație a discului este mult mai mică decât în cazul hard-diskului, de 300 r.p.m. (*rotații pe minut*). Doar mecanismul original de la firma Apple permitea viteze variabile de rotație, de 300 r.p.m sau 600 r.p.m. Tehnica vitezelor variabile de rotație a fost implementată apoi la unitățile de CD-ROM. Dischetele floppy de 1,44 MB sunt încă utilizate pentru transferul fişierelor de dimensiune mică de pe un calculator pe altul (când nu dispunem de o rețea de calculatoare) sau pentru stocare externă.

# 5

# REȚELE DE CALCULATOARE

## 5.1 Introducere

Dezvoltarea extraordinară în ultimii ani a industriei calculatoarelor şi a comunicațiilor a făcut ca în zilele noastre majoritatea calculatoarelor să fie conectate la o *rețea*. Practic nu mai putem să analizăm funcționarea unui calculator de sine-stătător ci se impune să avem o abordare din prisma rețelei de calculatoare la care acesta este conectat. Din această cauză vom prezenta în acest capitol câteva concepte, definiții şi elemente de bază utilizate în studiul rețelelor de calculatoare.

Definiția unanim acceptată de toate lumea (chiar dacă există mai multe definiții pentru termenul *rețea de calculatoare*) este aceea că o rețea de calculatoare (îi vom spune pe scurt, *rețea*) este o colecție de două sau mai multe calculatoare intreconectate între ele. Avantajul imediat al conectării la o rețea este evident: utilizatorii pot partaja fişiere şi dispozitive periferice (imprimante, modemuri, unități de backup, unități CD-ROM, etc.). În cazul în care rețele din mai multe locații sunt interconectate prin serviciile oferite de furnizorii de servicii Internet (linii telefonice, cablu, antene), utilizatorii pot folosi aplicații extrem de răspândite astăzi, precum cele de e-mail, chat, video-chat, video-conferința cu alți utilizatori conectați la rețea şi situați oriunde pe glob.

## 5.2 Topologii

Una dintre primele noțiuni prezentate atunci când se vorbeşte despre o rețea de calculatoare este noțiunea de topologie, care înseamnă, de fapt, structura rețelei. Topologia poate fi atât *fizică*, cât şi *logică*. O topologie

fizică reprezintă modalitatea de aşezare în spaţiu a componentelor reţelei, pe când topologia logică se referă la modalitatea de acces la mediu a dispozitivelor conectate la reţea.

### *5.2.1 Topologii fizice*

Cele mai răspândite topologii fizice sunt următoarele:

***Topologia de magistrală (bus)***

Această topologie foloseşte un singur cablu de conexiune principal iar dispozitivele din reţea sunt conectate toate la această conexiune (magistrală) (figura 5.1).

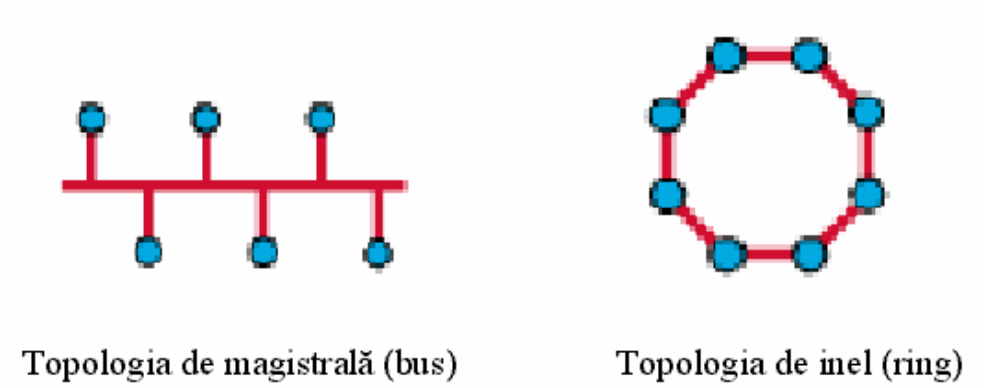



Fig. 5.1 Topologiile de magistrală şi inel

***Topologia de inel (ring)***

În această topologie fiecare dispozitiv este conectat de următorul, de la primul până la ultimul, ca într-un lanţ de interconectare (precum zalele unei brăţări) (figura 5.1).

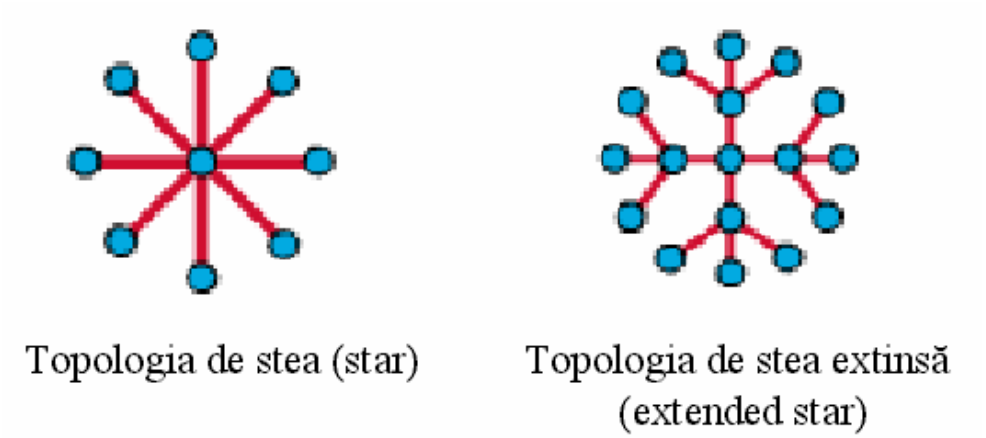



Fig. 5.2 Topologiile de stea şi stea extinsă

***Topologia de stea (star)***

În această topologie toate cablurile de conexiune ale dispozitivelor sunt legate la un dispozitiv central (denumit concentrator). De regulă, acest concentrator este un *hub* sau un *switch*. (figura 5.2).

***Topologia de stea extinsă (extended star)***

Topologia de stea extinsă uneşte topologiile stea intermediare la un concentrator central (hub sau switch) (figura 5.2).

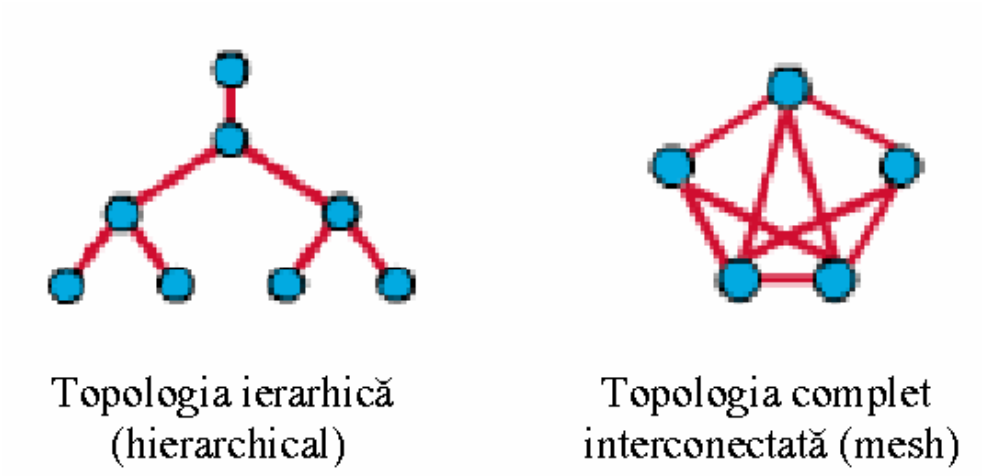



Fig. 5.3 Topologia ierarhică şi cea complet interconectată

***Topologia ierarhică (hierarchical)***

Această topologie este asemănătoare cu cea de stea extinsă, în care reţelele locale sub formă de stea sunt conectate la un computer ce controlează traficul în reţea (figura 5.3).

***Topologia complet interconectată (mesh)***

În această variantă topologică fiecare dispozitiv din reţea este interconectat cu toate celelalte (figura 5.3). În cazul în care numărul de gazde din reţea este ***n***, numărul conexiunilor necesare este ***n*(n-1)***!

### *5.2.2 Topologii logice*

În general există două tipuri de topologii logice (modalităţi de comunicaţie în reţea): topologia de tip *broadcast* şi cea de tip *token passing*.

#### *5.2.2.1 Topologia broadcast*

Cea mai cunoscută tehnologie broadcast este tehnologia Ethernet (apărută încă de la sfârşitul anilor 1970), larg utilizată în toate reţelele locale *LAN (Local Area Network)* din lume. Modalitatea de funcţionare a reţelelor Ethernet se bazează pe *CSMA/CD (Carrier Sense Multiple Access with*

*Collision Detection)*, un protocol de comunicație în rețea. Ideea de bază este simplă: o stație din rețeaua Ethernet poate trimite pachete de date în rețea atunci când nu există alte pachete de date care circulă în rețea (rețeaua este "liberă"). În caz contrar, stația care dorește să transmită pachete de date în rețea așteaptă până când rețeaua devine "liberă", exact ca o persoană care dorește să intre într-o conversație și așteaptă politicoasă ca altă persoană să termine de vorbit. Dacă mai multe stații din rețea încep să ransmită date în același timp, apare fenomenul de "coliziune". După apariția unei coloziuni, fiecare stație așteaptă un timp (de durată aleatoare) după care încearcă din nou să trimită pachetele de date. Dacă se întâmplă să se contorizeze 16 astfel de încercări nereușite de trimitere a pachetelor de date în rețea, aplicația care a generat operația de trimitere de date trebuie să reia cererea de transmitere de pachete de date în rețea.

Într-o rețea Ethernet, cu cât numărul utilizatorilor (stațiilor de lucru) crește, cu atât va crește și numărul coliziunilor, al erorilor și al cererilor de retransmisie, cu efect de multiplicare asemănător cu acela al "bulgărului de zăpadă". Fenomenul de apariție al coliziunilor este normal să se întâmple într-o rețea Ethernet, dar prea multe coliziuni vor duce la încetinirea funcționării rețelei. În cazul în care este utilizată mai mult de 50% din lățimea de bandă a rețelei, coliziunile încep să "gâtuiască", să încetinească rețeaua, producând fenomenul de congestie. Într-un astfel de moment, toate activitățile din rețea sunt încetinite: aplicațiile se deschid mai greu, transferul fișierelor durează mai mult, tipărirea la imprimanta de rețea se face mai greu și utilizatorii sunt obligați să aștepte. Dacă traficul de rețea depășește 60% din lățimea de bandă, deja rețeaua este supra-încărcată și există posibilitatea chiar ca rețeaua să "cadă". Standardul clasic Ethernet are asigură o lățime de bandă de 10 Mbps, în timp ce mai noile standarde *Fast Ethernet* (denumită și *100BaseT*) și *Gigabit Ethernet* (sau *1000BaseT*) funcționează în același mod cu Ethernet-ul clasic, dar la viteze de 100 Mbps, respectiv 1000 MBps.

#### *5.2.2.2 Topologia token passing*

Cea mai cunoscută tehnologie "*token passing"* este tehnologia *Token Ring*, construită ca alternativă la metoda Ethernet de detecție a coliziunilor. Ideea rețelei *Token Ring* este următoarea: în rețea circulă de la o stație la alta un *token (jeton de acces)* care "întreabă" stația respectivă dacă dorește sau nu să transmită pachete de date în rețea. Rețeaua Token

Ring are o topologie fizică de tip inel (după cum ne spune, de altfel, şi numele).

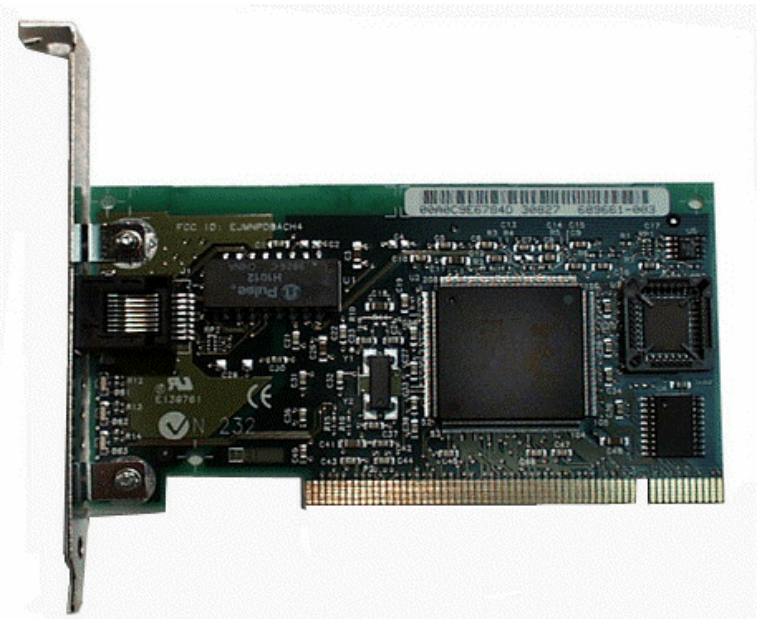


Fig. 5.4 Adaptor (placă) de rețea

În momentul în care o stație din rețea dorește să transmită date, preia token-ul şi îl converteşte într-un cadru de date pe care îl transmite în rețea. Acest cadru de date circulă în rețea, fiecare stație retransmite aceste date în jurul inelului iar stația destinație copiază acest cadru de date în memorie. În momentul în care cadrul trimis se întoarce înapoi la stația care l-a trimis, stația respectivă eliberează un nou token de acces în rețea.

Rețelele Token Ring operează la viteze de 4 sau 16 Mbps, dar costurile scăzute şi uşurința în exploatare a rețelelor Ethernet au făcut ca standardul Token Ring să fie din ce în ce mai puțin utilizat în noile implementări de rețea din zilele noastre.

### 5.3 Echipamente de rețea

Dispozitivele care sunt interconectate într-o rețea se numesc *gazde de rețea* sau, pe scurt, *gazde (hosts)*. Există o serie întreagă de dispozitive ce pot exista într-o rețea, dar elementele *de bază* într-o rețea simplă sunt

următoarele: cel puțin două calculatoare, o interfață de rețea (placă de rețea) instalată pe fiecare calculator în parte, un mediu de conexiune (tipic un cablu, dar există şi conexiuni fără fir - *wireless*) şi un sistem de operare de rețea (Unix, Linux, Windows 95, 98, 2000, XP, NT sau Novell Netware). De asemenea, majoritatea rețelelor posedă şi un *hub* sau un *switch* pentru interconectarea gazdelor din rețea. Prezentăm în continuare pe scurt echipamentele de bază dintr-o rețea.

Fig. 5.5 Placă de rețea PCMCIA pentru laptop

***Placa de rețea***

Placa de rețea (figura 5.4), denumită şi interfață de rețea sau adaptor de rețea sau *NIC (Network Interface Card),* reprezintă un element de bază într-o rețea de calculatoare, având în vedere că orice calculator conectat la rețea trebuie să posede un astfel de adaptor. Placa de rețea este o placă cu circuite electronice sau o placă PCMCIA (pentru laptop – figura 5.5) şi are rolul de a asigura conexiunea la rețea a unui echipament (calculator, imprimantă etc.). Fiecare placă de rețea are o adresă unică, numită şi adresă fizică sau adresă *MAC (Media Access Control)*.

Atunci când se alege o placă de rețea pentru un calculator sau pentru alt dispozitiv pe care vrem să îl conectăm la o rețea trebuie să luăm în considerare următoarele:

- Tipul rețelei (Ethernet, Token Ring, etc.);
- Tipul mediului de conexiune (cablu coaxial, fibră optică, cablu UTP sau legătură wireless);
- Tipul magistralei de sistem (PCI, ISA, etc.).

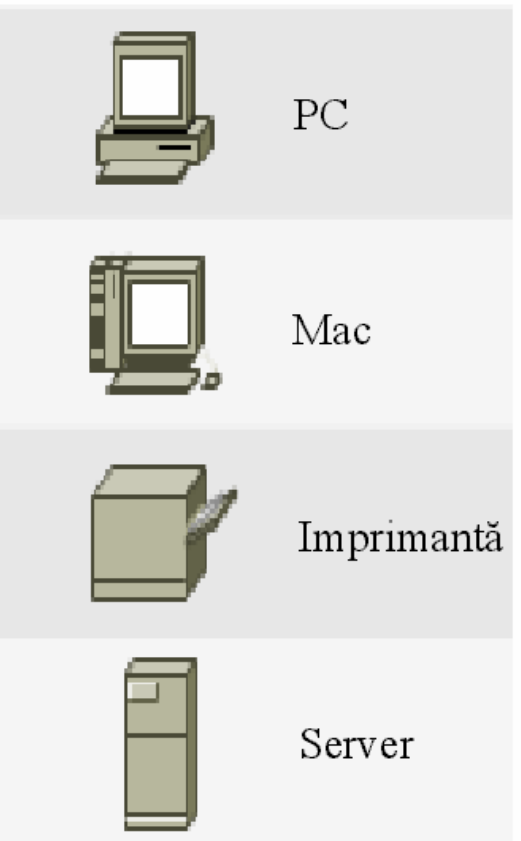



Fig. 5.6 Echipamente gazdă într-o rețea

***Echipamente gazdă într-o rețea***

Toate calculatoarele, indiferent de tipul lor, imprimantele sau serverele conectate la o rețea se numesc gazde (figura 5.6).

***Repetorul (repeater)***

Un echipament foarte simplu întâlnit în rețele este *repetorul (repeater-ul),* dispozitiv care nu face altceva decât să amplifice şi să regenereze semnale la nivel de biți de-a lungul rețelei. Aceste echipamente se întâlnesc atunci când se doreşte extinderea dimensiunii maxime pentru tipul de cablu respectiv şi prezintă din punct de vedere constructiv un port de intrare şi un port de ieşire.

***Hub-ul***

Hub-ul acționează tot la nivel de bit şi de regenerare a semnalelor, fiind denumit, de fapt, repetor multiport, deoarece prezintă un port de intrare, dar mai multe porturi de ieşire. Introducerea unui hub în rețea crează un punct de conexiune centrală (la care sunt legate toate gazdele) şi măreşte fiabilitatea. Hub-urile pot fi de mai multe tipuri, unele fiind *hub-uri active*, ce folosesc energie electrică pentru a regenera semnalele, în timp alte hub-uri sunt *hub-uri pasive* care doar împart semnalele la mai mulți utilizatori. Hub-urile *pasive* nu regenerează biții, nu extind lungimea cablării şi nu permit ca două sau mai multe gazde să fie conectate pe acelaşi segment de cablu. Există, de asemenea, şi *hub-uri inteligente*, care pot fi programate pentru a se administra traficul în rețea.

***Bridge-ul***

Bridge-ul acționează la un nivel superior (nivelul 2 din modelul OSI, model pe care îl vom prezenta în secțiunea 5.4.3.6), conectând două segmente de rețea. De asemenea, un bridge filtrează traficul în rețea pe baza adreselor MAC, regula generală fiind următoarea: traficul local (ce provine din acelaşi segment de rețea) este menținut local iar traficul din afara segmentului de rețea este direcționat în afara segmentului de rețea. Un bridge creează şi administrează nişte tabele de adrese MAC pe baza cărora poate face filtrarea traficului după regula de mai sus.

***Switch-ul***

Switch-ul este tot un echipament de rețea ce acționează la nivelul 2 din modelul OSI, fiind denumit şi *bridge multiport*. Deciziile de direcționare a traficului din rețea se fac tot pe baza adreselor MAC, comutarea semnalelor transmise făcându-se către portul unde este conectat calculatorul destinație (cel căruia i se adresează pachetul de date). Switch-urile arată oarecum asemănător cu un hub dar administrează transmisiunile de pachete de date mult mai eficient, combinând capacitatea de conectivitate a hub-ului cu reglarea traficului din rețea (pe care o face un *bridge*) la nivelul fiecărui port de conexiune.

***Router-ul***

Router-ul este un echipament de rețea mai complex (şi mai scump, de asemenea) ce acționează la nivelul 3 din modelul OSI. Acest dispozitiv poate lua decizii de direcționare a pachetelor de date din rețea pe bază de clase de adrese logice (adrese IP), putând conecta diferite tehnologii de nivel 2 OSI, cum ar fi Ethernet, FDDI, Token Ring etc. Routerele reprezintă elementele centrale ale Internetului, ele examinează pachetele de date de intrare şi le direcționează către interfețele respective de ieşire. Un router este asemănător cu un calculator, având procesor, memorie, interfețe de intrare/ieşire, având deci putere de procesare şi de luare a unei decizii de direcționare a unui pachet de date către o anumită interfață pe baza unor aşa numite *protocoale de rutare*.

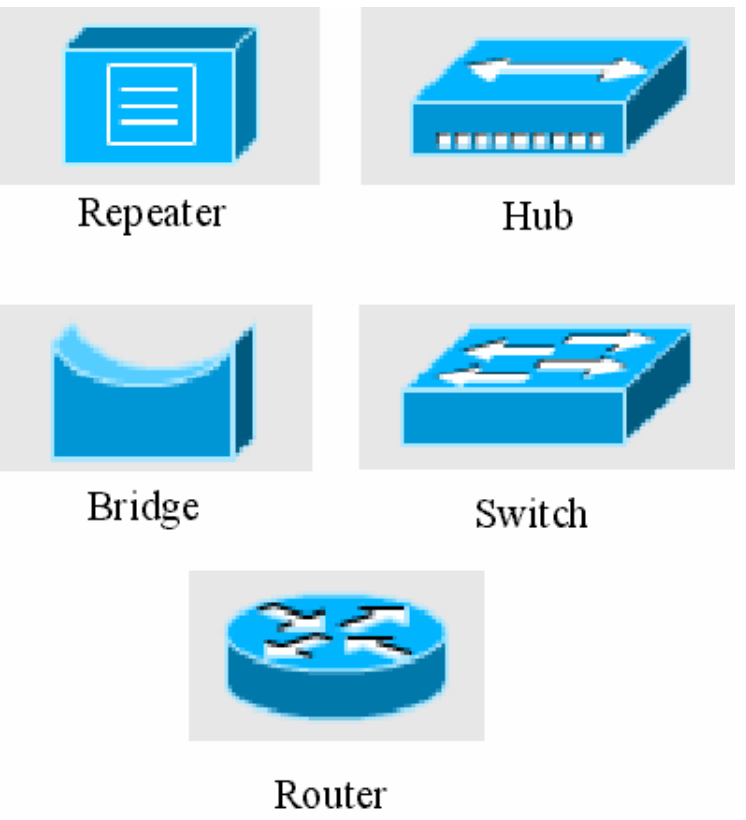



Fig. 5.7 Simbolurile grafice utilizate pentru repetor, hub, bridge, switch, ruter

Simbolurile grafice utilizate pentru realizarea diagramelor de rețea sunt prezentate în figura 5.7.

## 5.4 Modele utilizate în dezvoltarea şi studiul rețelelor de calculatoare

### *5.4.1 Introducere*

Primele rețele de calculatoare din lume erau formate, de regulă, din calculatoare ce proveneau de la acelaşi producător, neexistând posibilitatea de a face să coopereze computere şi echipamente de rețea produse de firme diferite. Pe măsură ce numărul de calculatoare a crescut şi complexitatea rețelelor s-a mărit, a apărut necesitatea de a putea fi integrate împreună soluții provenite de la mai mulți fabricanți de computere şi tehnologii de rețea. La sfârşitul anilor 1970, *Organizația Internațională pentru Standardizare (ISO – International Organization for Standardization)* a început dezvoltarea primului model arhitectural – denumit ***OSI*** *(Open Systems Interconnection)* pe baza căruia să se rezolve această necesitate.

Acest model s-a impus ca un standard bine cunoscut în lumea rețelelor de calculatoare, iar pentru că organizația *ISO* a conceput acest model, numele complet al său este modelul ***ISO-OSI***. Acest model doreşte să fie un ajutor pentru ca producătorii de calculatoare şi de echipamente de rețea să aibă produse *interoperabile* cu cele similare ale altor producători. Modelul ISO-OSI este modelul arhitectural de bază al rețelelor de calculatoare, descriind modul în care aplicațiile de pe un computer comunică prin intermediul mediilor de rețea cu aplicațiilor de pe un alt computer aflat în rețea.

Alte două modele arhitecturale de rețea pe care le vom prezenta în continuare sunt modelul TCP/IP şi modelul ierarhic Cisco. Toate aceste modele au o caracteristică principală comună, în sensul că abordarea problematicii rețelistice se face pe *nivele*.

### *5.4.2 Necesitatea unui model ierarhic*

Pentru a înţelege mai bine de ce a apărut necesitatea existenţei unui model după care să fie proiectate, dezvoltate, analizate şi depanate reţelele de calculatoare trebuie să definim noţiunea de *flux informaţional*. Considerând exemplul a două calculatoare aflate într-o reţea (figura 5.8), *comunicarea* dintre acestea se face pe baza unui schimb de date; această deplasare a datelor de la calculatorul sursă la cel destinaţie poartă numele de *flux de date* sau, pe scurt, *flux*.

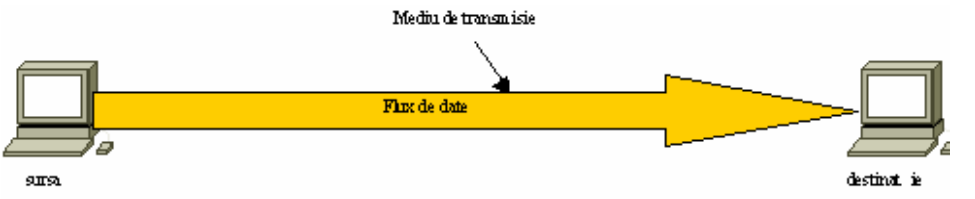



Fig. 5.8 Fluxul de date între sursă şi destinaţie

Putem face analogii ale scurgerii fluxului de date cu numeroase exemple din viaţa de zi cu zi: traficul maşinilor pe stradă, scurgerea apei într-o conductă sau drumul parcurs de o scrisoare de la expeditor la destinatar. În toate aceste exemple este vorba de o mişcare a unor obiecte (fie că este vorba de maşini, apă sau scrisori) dintr-un loc în altul, iar această mişcare reprezintă un *flux*. În legătură cu fluxul de date dintr-o reţea de calculatoare, apar o serie de întrebări care trebuie lămurite:

- Care este fluxul?
- Care sunt diferitele forme de flux?
- Ce reguli guvernează acest flux?
- Unde apare acest flux?

Pentru a clarifica răspunsurile la aceste întrebări ce apar când este forma de fluxul de date dintr-o reţea de calculatoare s-a recurs la divizarea problemei de comunicaţie pe mai multe nivele, lucru pe care îl fac şi modelele arhitecturale de reţea. Divizarea chestiunii comunicaţiei pe mai multe nivele are următoarele avantaje:

- Se împarte problema comunicaţiei din reţea în piese mai mici şi mai simple, deci mai uşor de analizat;

- Se doreşte standardizarea componentele de rețea pentru a putea permite dezvoltare şi suport multiproducător;
- Permite diferitor tipuri de hardware şi software din rețea să comunice între ele;
- Este o modalitate de prevenire a faptului că o modificare ce apare la un nivel să afecteze celelalte nivele, astfel încât să se dezvolte mai rapid.

### *5.4.3 Modelul ISO-OSI*

Lansat oficial în 1984, modelul ISO-OSI reprezintă modelul arhitectural principal pe baza căruia rețelele de calculatoare sunt proiectate, analizate, dezvoltate, implementate sau depanate. Acest model este conceput să trateze rețelele de calculatoare pe mai multe nivele, făcând astfel ca problemele comunicației (fluxurile din rețea) să fie divizate în probleme mai simple şi mai uşor de analizat, corespunzătoare unui nivel din rețea. Cu ajutorul modelului OSI se îmbunătățeşte transferul datelor dintre nodurile unei rețele, având în vedere că una dintre caracteristicile sale principale este aceea de a asista modalitatea de transfer a datelor între două sisteme terminale din rețea.

| | |
|---|---|
| **APLICAȚIE** | Interfața cu utilizatorul |
| **PREZENTARE** | Modalitatea de prezentare a datelor |
| **SESIUNE** | Separă datele diferitelor aplicații |
| **TRANSPORT** | Asigură livrarea datelor la destinație<br>Asigură corecția datelor înaintea transmiterii |
| **REȚEA** | Se ocupă cu adresarea logică pe care ruterele o utilizează pentru determinarea rutei până la destinație |
| **LEGĂTURĂ DE DATE** | Pachetele de date sunt transformate în octeți şi octeții în cadre<br>Oferă acces la mediu prin utilizarea adreselor MAC<br>Asigură detecția erorilor |
| **FIZIC** | Mută şiruri de biți între echipamente<br>Specifică tipul de cablare, viteza de transmisie, voltaje, tipuri de conectori, etc. |

Fig. 5.9 Nivelele modelului OSI

Modelul OSI este practic un set de principii de bază pe care dezvoltatorii de aplicaţii de reţea îl pot folosi pentru a crea şi implementa aceste aplicaţii. De asemenea, modelul oferă cadrul specific pentru *crearea şi implementarea standardelor de reţea, a echipamentelor şi a schemelor de interconectare în reţea*. Modelul OSI descrie modalitatea în care datele şi informaţiile din reţea sunt transmise de la o aplicaţie de pe un computer către o altă aplicaţie de pe alt computer; acest lucru se face folosind o abordare pe 7 nivele. Cele 7 nivele ale modelului OSI sunt împărţite în două grupuri. Primul grup, format din cele trei nivele superioare defineşte modul de comunicare între aplicaţiile de pe staţiile terminale din reţea şi modul de comunicare cu utilizatorii. Cel de-al doilea grup, format din cele 4 nivele inferioare defineşte modul de transmitere a datelor de la o sursă la o destinaţie. În figura 5.9 sunt prezentate cele 7 nivele ale modelului OSI împreună cu câteva caracteristici principale ale fiecărui nivel.

Modalitatea de funcţionare a nivelelor din modelul OSI este următoarea: fiecare nivel oferă servicii nivelului imediat superior (nivelul fizic oferă servicii nivelului legătură de date, nivelul legătură de date oferă servicii nivelului reţea ş.a.m.d.), excepţie făcând nivelul aplicaţie care nu are un alt nivel superior. Comunicarea între două sisteme terminale din reţea se face, de asemenea, pe baza unor protocoale corespunzătoare nivelelor din modelul OSI la care acestea activează. Astfel, nivelul aplicaţie al uni sistem terminal comunică cu nivelul aplicaţie al celuilalt sistem, nivelul reţea cu nivelul reţea etc.

Vom face în continuare o analiză mai detaliată pentru fiecare nivel din modelul OSI în parte, pe baza modelului top-down, plecând de la nivelul aplicaţie (de vârf) şi ajungând la nivelul fizic (de bază).

### *5.4.3.1 Nivelul aplicaţie*

Nivelul aplicaţie din modelul OSI este folosit de către utilizatori pentru asigurarea interfeţei cu calculatorul. Acest nivel se ocupă cu identificarea şi stabilirea partenerului de comunicaţie şi determină dacă există suficiente resurse pentru stabilirea comunicaţiei dorite. În timp ce unele aplicaţii necesită numai resurse locale, alte aplicaţii pot necesita componente ce provin de la mai multe aplicaţii de reţea – exemple pot fi: transferuri de fişiere, localizarea de informaţii, poşta electronică (e-mail), acces la distanţă, management de reţea etc.

Dacă în trecut comunicația dintre aplicații se facea de regulă la nivelul unei organizații sau firme, există o tendință generală în ultimii ani de a mări spectrul acestor comunicații la nivel interorganizațional, dezvoltându-se aplicații interrețea precum:

- WWW (World Wide Web) – sau prescurtat *Web*, asigură conexiunea dintre servere din întreaga lume folosind diverse formate. Majoritatea formatelor sunt multimedia şi includ text, grafică, sunet sau video. Programele de navigare Web (denumite *browsere*) precum Netscape Navigator, Internet Explorer, Opera şi altele ne ajută să accesăm şi să vizualizăm *site*-urile Web.
- EDI (Electronic Data Interchange) – reprezintă o colecție de standarde şi procese ce facilitează fluxul de operații ce includ contabilizarea, transportul/recepția, inventarierea datelor interschimbate între diferite companii.
- Utilitare pentru navigare Internet – acestea includ aplicații precum Gopher, WAIS sau motoare de căutare ca Yahoo!, Lycos, Alta Vista, Google, Excite care ajută utilizatorii să localizeze informațiile dorite.
- Servicii pentru tranzacții financiare – acestea colectează şi oferă informații referitoare la investiții, piețe financiare, active financiare, rate de schimb valutar, etc. pentru clienții permaneți sau ocazionali.
- Sisteme BBS (Bulletin Board System) – acestea sunt baze de date de produse software unde putem găsi o mulțime extrem de mare de programe din diverse domenii. Software-ul existent aici este, de regulă, gratis sau promoțional, adică programele nu funcționează cu toate facilitățile până când nu sunt cumpărate (de tip *shareware*).

#### *5.4.3.2 Nivelul prezentare*

După cum spune şi numele său, acest nivel oferă o modalitate de prezentare a datelor nivelului superior, nivelului aplicație. Nivelul prezentare acționează ca un translator oferind funcții de codificare şi conversie. Modalitatea de a asigura o transmisie cu succes a datelor este aceea de a codifica datele într-un format cunoscut înaintea începerii transmisiunii. Acest lucru se face deoarece computerele sunt astfel construite pentru a recepționa date formatate generic şi a le converti într-un

format nativ pentru citire (de exemplu folosind codul ASCII, EBCDIC sau Unicode). Prin oferirea de servicii de translatare, nivelul prezentare asigură faptul că datele transmise de nivelul aplicație al unui sistem terminal sunt înțelese de nivelul aplicație al altui sistem terminal.

Modelul OSI oferă o serie de protocoale standard pentru acest nivel, protocoale ce definesc modalități de formatare a datelor. Operații precum criptarea, decriptarea, compresia sau decompresia datelor sunt în strânsă legătură cu acest nivel. Majoritatea standardelor acestui nivel se referă la operații multimedia. Exemple de astfel de standarde sunt:

- *GIF (Graphic Interchange Format)* – este un format pentru fişiere de tip grafic;
- *PICT (PICTure)* – acest format grafic este folosit de programe Macintosh sau PowerPC pentru transferul de grafice QuickDraw;
- *TIFF (Tagged Image File Format)* – este un format standard grafic pentru imagini de înaltă rezoluție;
- *JPEG (Joint Photographic Expert Group)* – este un alt format grafic;
- *MIDI (Musical Instrument Digital Interface)* este folosit pentru crearea de fişiere ce conțin muzică în format digital;
- *AVI (Audio Video Interleave)* este un format pentru fişiere de sunet şi video dezvoltat de Microsoft. El reprezintă formatul standard pentru PC-uri, făcând parte integrantă din distribuția "Video for Windows (VfW)" a sistemului de operare Windows;
- *MPEG (Moving Picture Experts Group)* – este un standard pentru compresia şi codificarea fişierelor ce conțin imagini în mişcare – oferă rate de transfer de până la 1,5 Mbps;
- *QuickTime* – este un alt format popular pentru fişiere ce conțin date audio/video – dezvoltat inițial pe platforme Macintosh şi PowerPC.

#### *5.4.3.3 Nivelul sesiune*

Acest nivel este responsabil cu inițierea, administrarea şi încheierea sesiunilor de comunicație între entitățile nivelului prezentare. De asemenea, nivelul sesiune oferă un control al dialogului între nodurile terminale şi

coordonează comunicația între sisteme. Există trei moduri de comunicație între sistemele terminale: *simplex*, *half-duplex* şi *full-duplex*•. Exemple de protocoale ale nivelului sesiune:

- NFS (Network File System) – este un sistem de fişiere de rețea dezvoltat de Sun Microsystems şi folosit împreună cu Unix şi TCP/IP pentru a permite accesul transparent la resurse din rețea;
- X Window – este o interfață grafică extrem de folosită de către terminalele inteligente pentru comunicarea cu computere Unix din rețea, permițând operarea ca şi cum ar fi monitoare locale ataşate;
- SQL (Structured Query Language) – dezvoltat de compania IBM pentru a oferi utilizatorilor o modalitate simplă şi standardizată de a îşi specifica necesitățile pentru informații aflate pe stația de lucru locală sau în altă parte în rețea;
- RPC (Remote Procedure Call) reprezintă un instrument client/server folosit în medii de rețea. Procedurile sunt create pe clienți şi operează pe servere;
- AppleTalk Session Protocol (ASP) este un alt mecanism client/server care stabileşte şi administrează sesiuni de comunicație între un client AppleTalk şi un server.

#### *5.4.3.4 Nivelul transport*

Nivelul transport oferă servicii care segmentează şi reasamblează datele provenite de la aplicații de nivel superior, unificându-le într-un flux de date. De asemenea, acest nivel din modelul OSI asigură servicii de transport între sursă şi destinație, putând stabili o conexiune logică între sistemul transmițător şi sistemul destinație dintr-o rețea. Exemple de protocoale folosite la acest nivel sunt TCP (Transmission Control Protocol) şi UDP (User Datagram Protocol).

---

• Modul *simplex* implică faptul că transmisia de date se face într-un singur sens între stația sursă şi cea destinație. Modul *half-duplex* oferă posibilitatea transmisiei în ambele sensuri între două sisteme terminale, dar într-un singur sens la un moment dat, în timp ce modul *full-duplex* semnifică faptul că transmisia se poate efectua în ambele sensuri la acelaşi moment dat.

Nivelul transport este responsabil cu asigurarea mecanismelor pentru multiplexarea datelor provenite de la nivelele superioare, stabilirea sesiunilor de comunicaţie şi stoparea circuitelor virtuale. Noţiunile fundamentale ce trebuie luate în considerare la acest nivel sunt: *controlul fluxului, comunicaţii orientate pe conexiune, windowing* sau *transmisia de confirmări*.

***Controlul fluxului***

Una dintre sarcinile de bază ale nivelului transport o constituie asigurarea integrităţii datelor. Acest lucru este realizat prin controlul fluxului de date şi prin asigurarea unui mecanism fiabil de transport de date între un sistem sursă şi un sistem destinaţie. Controlul fluxului împiedică faptul ca un sistem transmiţător să transmită mai multe date decât sistemul receptor poate primi, deoarece un astfel de eveniment poate duce la pierderi de date. Transportul fiabil al datelor între sursă şi destinaţie implică o sesiune de comunicaţie orientată pe conexiune iar protocoalele implicate în acest fenomen vor asigura următoarele:

- Transmiţătorul va primi o confirmare din partea receptorului pentru segmentele de date transmise şi recepţionate;
- Toate segmentele care nu primesc această confirmare vor fi retransmise;
- Segmentele ajunse la destinatar vor fi aranjate în ordinea corespunzătoare transmisiei, indiferent de ordinea la care au ajuns la destinaţie;
- Se va asigura un control al fluxului de date pentru a evita congestiile, supraîncărcarea sau pierderea de date.

***Comunicaţii orientate pe conexiune***

Modalitatea de lucru a unei comunicaţii orientate pe conexiune este următoarea: pentru a se asigura un transport fiabil de date de la sursă la destinaţie, sistemul care trimite date stabileşte într-o primă etapă o sesiune *orientată pe conexiune* cu sistemul său pereche cu care comunică. În acest caz, programele aplicaţie de pe cele două sisteme terminale încep prin a trimite un semnal sistemelor de operare care anunţă faptul că se va iniţia o conexiune. Cele două sisteme de operare comunică între ele prin trimiterea de mesaje de-a lungul reţelei confirmând că se aprobă transferul de date şi că acesta poate începe. Din momentul realizării sincronizării între sisteme, se poate spune că a fost creată o conexiune şi transferul de date se poate

inițiа. Un astfel de procedeu se mai numește *three-way-handshake,* adică stabilirea conexiunii în trei etape (figura 5.10).

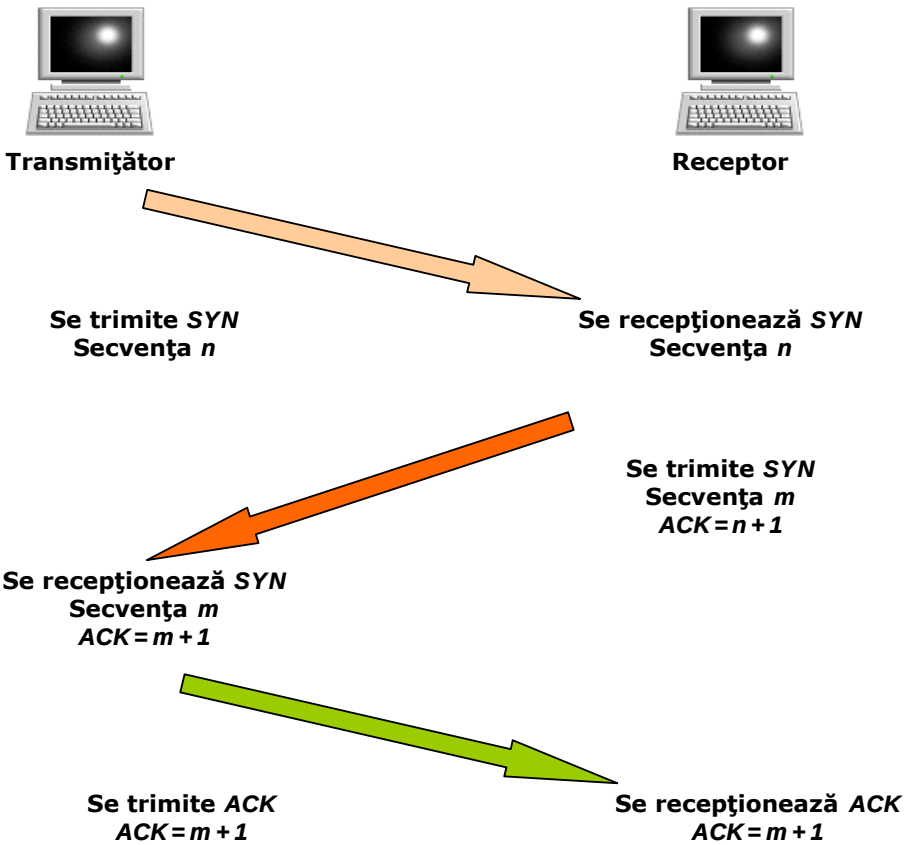



Fig. 5.10 Comunicația de tip *three-way-handshake*

Detaliat, conexiunea de tip *three-way-handshake* presupune următoarele etape: la început, stația transmițătoare inițiază (trimite un apel ***SYN***) conexiunea prin trimiterea unui pachet de date ce are un număr de secvență ***n*** şi care conține un bit în *header* care indică faptul că secvența reprezintă o cerere de conexiune. În faza a doua, stația receptoare primește pachetul, înregistrează secvența ***n*** şi răspunde cu o confirmare (***ACK****nowledgement*) ***n+1*** şi include secvența inițială proprie ce are numărul m. Confirmarea cu numărul ***n+1*** semnifică faptul că stația receptoare a primit toți octeții de date până la ***n*** şi aşteaptă secvența următoare ***n+1***.

Din momentul iniţierii comunicaţiei, cele două sisteme terminale comunică prin intermediul unui protocol anume pentru a asigura transferul corect al datelor pe toată durata transmisiunii.

O altă tehnică obişnuită de transmitere fiabilă a datelor este *PAR (Positive Acknowledgement and Retransmission)* în care staţia sursă trimite un pachet de date, porneşte un contor de timp şi aşteaptă o confirmare înainte de a trimite următorul pachet. Dacă timpul scurs până la primirea confirmării depăşeşte valoarea specificată, staţia sursă retransmite acel pachet de date şi porneşte din nou contorul de timp.

În timpul unui transfer de date poate apărea la un moment dat o *congestie*. Acest lucru se întâmplă atunci când un computer transmite date cu o mare viteză generând un trafic prea rapid pentru reţea sau atunci când mai multe computere transmit simultan date prin intermediul unei singure porţi *(gateway)* către destinaţie. În acest din urmă caz, fie poarta de transmitere fie destinaţia pot deveni aglomerate iar cauza problemei nu provine de la un singur sistem terminal. Congestia în reţea este similară cu traficul de maşini de pe o autostradă: un număr prea mare de maşini ce utilizează autostrada conduce la un trafic aglomerat şi la apariţia de gâtuiri ale traficului (aşa numitele ambuteiaje). În momentul în care o staţie din reţea recepţionează *datagrame* (structuri de date) prea rapid pentru a le putea procesa, acestea sunt stocate într-o zonă de memorie tampon numită *buffer*. Această modalitate de a copia datagramele în memoria tampon rezolvă problema doar dacă acest trafic supraaglomerat este de scurtă durată. Dacă supraîncărcarea persistă, memoria tampon va deveni insuficientă pentru stocarea tuturor datagramelor primite şi datagramele ulterioare primite se vor pierde. Tocmai pentru a se evita astfel de pierderi de date a fost creată funcţia de *transport* ce controlează fluiditatea traficului de date din reţea.

Un alt procedeu utilizat pentru transmiterea datelor de la o sursă la o destinaţie se numeşte *windowing*. Deoarece viteza efectivă de transfer a datelor ar fi prea înceată dacă staţia care transmite ar aştepta o confirmare pentru fiecare segment de date trimis, după iniţierea comunicaţiei se pot transmite mai multe pachete de date înainte de a primi o confirmare din partea destinatarului. Numărul de segmente transmise până la primirea unei confirmări poartă numele de *window* (îi vom spune *fereastră de date*). Există mai multe tipuri de folosire a acestei tehnici: în timp ce unele protocoale cuantifică informaţia trimisă într-o fereastră de date ca multiplu de pachete, alte protocoale (între care şi TCP/IP) folosesc multiplii de octeţi pentru ferestrele de date. Figura 5.11 ilustrează acest procedeu folosind

două dimensiuni window: 1 şi 3. În cazul unui window de mărime 1, transmițătorul aşteaptă o confirmare pentru fiecare segment de date transmis înainte de a trimite alt segment de date. În cazul unui window de dimensiune 3, se vor transmite 3 pachete de date fără a se aştepta confirmarea.

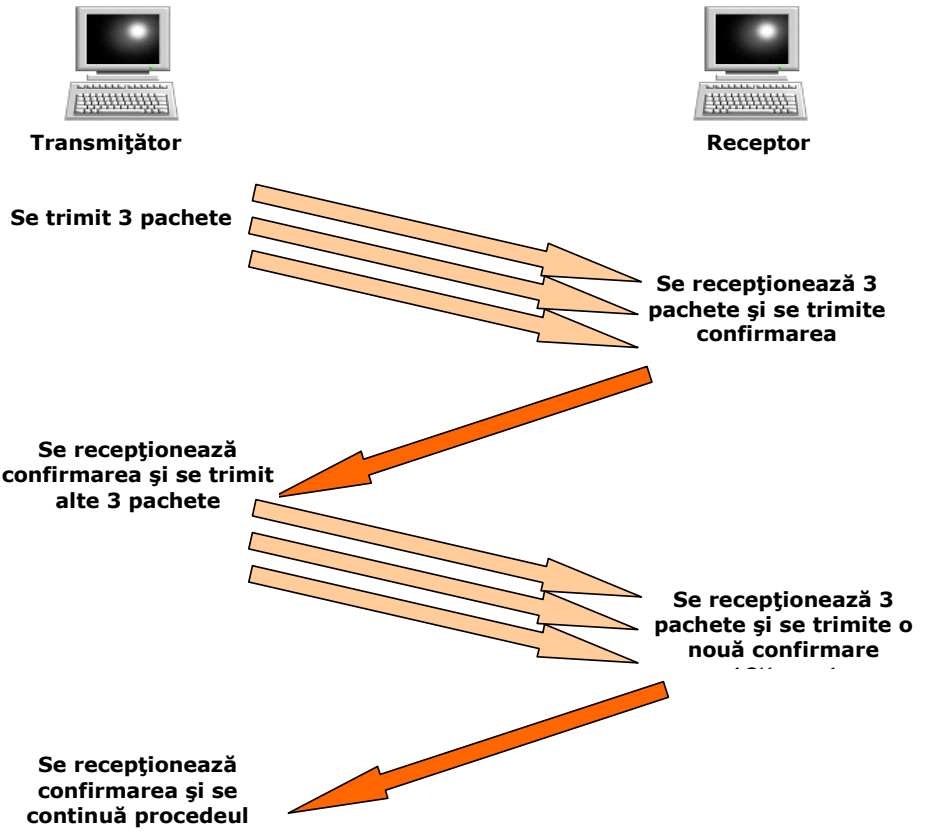

Transmiţător
Receptor
Se trimit 3 pachete
Se recepţionează 3 pachete şi se trimite confirmarea
Se recepţionează confirmarea şi se trimit alte 3 pachete
Se recepţionează 3 pachete şi se trimite o nouă confirmare
Se recepţionează confirmarea şi se continuă procedeul

Fig. 5.11 Ilustrarea procedeului *windowing* cu o "*fereastră"* de dimensiune 3

Protocolul TCP utilizează o astfel de tehnică de *windowing* denumită *sliding-window (fereastră glisantă)*. Denumirea semnifică faptul că dimensiunea ferestrei de date "glisează", adică este negociată dinamic în timpul transmisiunii.

### *5.4.3.5 Nivelul rețea*

Nivelul rețea din modelui OSI este responsabil cu direcționarea (rutarea) datelor între rețele şi cu adresarea interrețea. Acest lucru înseamnă că nivelul rețea are grijă de datele transmise între echipamente care nu sunt în aceeaşi rețea locală. Dispozitive speciale numite *rutere* funcționează la acest nivel OSI oferind servicii de rutare într-o interrețea. În funcție de nivelul OSI la care activează, spunem că echipamentul respectiv este un echipament de nivel corespunzător; astfel, ruterele se mai numesc şi *echipamente de nivel 3*.

Vom prezenta în continuare cum funcționează mecanismul de *rutare*, adică modalitatea prin care ruterele direcționează pachetele de date spre destinație. Porturile de comunicație în rețea ale unui ruter se numesc *interfețe*. Primul lucru care se face atunci când un pachet de date ajunge la o interfață a unui ruter este să se verifice adresa destinație (adresa *IP -Internet Protocol*). Dacă acest pachet nu are ca adresă de destinație chiar ruterul, atunci ruterul consultă un tabel de adrese de rețea ce se numeşte *tabelă de rutare (routing table)*. În continuare pachetul se trimite la o interfață de ieşire a ruterului pentru a fi descompus în cadre şi trimis în rețeaua locală. În situația în care adresa destinație nu se regăseşte în tabela de rutare atunci pachetul este "aruncat".

La nivelul rețea OSI se utilizează două tipuri de pachete: pachete ce conțin date efective şi pachete ce conțin informații de reîmprospătare a rutelor de rețea.

- *Pachetele de date* sunt folosite pentru a transporta datele utilizatorilor în cadrul inter-rețelei iar protocoalele folosite pentru acest tip de trafic se numesc *protocoale rutate (routed protocols),* categorie din care fac parte şi protocoalele IP şi IPX.
- *Pachetele ce conțin informații de reîmprospătare a rutelor* sunt utilizate pentru a aduce la zi rutele vecine provenite de la ruterele conectate în inter-rețea. Protocoalele utilizate în acest sens se numesc *protocoale de rutare (routing protocols)* şi ca exemple pot fi date RIP (Routing Information Protocol – cel mai utilizat protocol de rutare din Internet), EIGRP (Enhanced Interior Gateway Routing Protocol) sau OSPF (Open Shortest Paths First).

Tabelele de rutare memorate şi folosite de către rutere conțin următoarele informații: *adrese de rețea, denumiri de interfețe şi metrică.*

Adresele de rețea sunt specifice protocoalelor folosite; un ruter trebuie să administreze o tabelă de rutare pentru fiecare protocol în parte deoarece acestea gestionează rețelele folosind scheme de adresare diferite. Putem compara această situație cu aceea a incripționării denumirii unei străzi în mai multe limbi care sunt vorbite de către locuitorii de pe acea stradă. Denumirile de interfață specifică interfața de ieşire pe care pachetul o va urma atunci când este adresat unei anumite rețele.

*Metrica* reprezintă distanța către rețeaua destinație, distanță care se calculează în funcție protocolul de rutare utilizat. În cazul protocoalelor bazate pe vectorul-distanță *metrica (hop-count)* este de numărul de rutere intermediare existente până la acea rețea. Alte tipuri de metrice includ calculul lățimii de bandă, întârzierile sau un contor de timp (*tick-count*, cu o valoare de *1/18 dintr-o secundă*).

Cel mai utilizat protocol pentru transferul informațiilor de rutare între rutere direct conectate este RIP. Acest protocol interior de rutare (IGP – Interior Gateway Protocol) calculează distanțele până la destinație, permițând ruterelor să-şi reîmprospăteze tabelele de rutare la intervale programabile, de regulă la fiecare 30 de secunde. Dezavantajul acestui procedeu este acela că se generează trafic suplimentar în rețea. Protocolul RIP oferă posibilitatea ruterului de a alege drumul (ruta) din rețea pe care se vor transmite datele pe baza *vectorului distanță (distance-vector)*. Ori de câte ori sunt transmise date în rețea ce trec printr-un nou ruter se consideră că "s-a trecut un hop". Spre exemplu, dacă până la destinație un pachet trece prin 3 rutere intermediare (*hopuri*) atunci avem de-a face cu o metrică în valoare de 3.

În cazul în care avem mai multe rute până la destinație, protocolul RIP selectează ruta cu numărul minim de hopuri, care însă nu este neapărat cea mai rapidă rută. Cu toate acestea, RIP rămâne foarte popular şi răspândit printre protocoalele de rutare folosite. O altă problemă a protocolului RIP este aceea că acesta nu poate alege rute ce au numărul de hopuri până la destinație mai mare de 15. Din această cauză pot apărea cazuri în care destinația se află prea departe şi atunci se va considera că nu poate fi atinsă.

Ruterele sunt dispozitive ce nu iau în considerare domeniile de tip broadcast. O transmisie de tip *broadcast* în rețea semnifică faptul că are ca destinație toate dispozitivele din acea rețea; din această cauză, dacă transmisiunile de tip broadcast sunt prea frecvente va rezulta o supra-încărcare a traficului rețelei. Ruterele delimitează atât domenii de broadcast cât şi domenii de coliziuni, lucru pe care îl fac însă şi dispozitivele nivelui 2 din modelul OSI, nivelul legătură de date. Fiecare interfață a ruterului

reprezintă o rețea separată şi trebuie să i se atribuie un număr de identificare unic. În acest caz fiecare sistem terminal conectat la ruter trebuie să utilizeze acelaşi număr de rețea. Caracteristicile cele mai importante ale ruterelor sunt următoarele:

- Ruterele folosesc adrese logice în header-ul nivelului rețea pentru a determina ruta pe care să direcționeze pachetul de date spre destinație;
- Ruterele nu direcționează pachete de tip *broadcast* sau *multicast* (pachete ce au mai multe destinații);
- Ruterele pot folosi liste de acces setate de administratorul rețelei pentru a asigura securitatea rețelei şi a controla pachetele de date care intră sau ies din ruter;
- Ruterele pot asigura simultan atât servicii de nivel 2 de tip bridging cât şi servicii de rutare de nivel 3;
- Echipamentele de nivel 3 (în acest caz ruterele, dar pot exista şi *switch-uri* cu funcții de nivel 3) pot oferi conexiuni între rețele virtuale LAN (Virtual LAN).

#### *5.4.3.6 Nivelul legăturii de date*

Nivelul legăturii de date *(data link)* din modelul OSI are două funcțiuni principale: asigură faptul că mesajele sunt transmise dispozitivului corespunzător şi translatează aceste mesaje provenite de la nivelul superior (nivelul rețea) în biți pe care îi transmite mai departe nivelului fizic. Mesajele sunt formatate în *cadre de date (data frames)* şi li se adaugă un header ce conține adresele hardware pentru destinație şi pentru sursă. Aceste informații adiționale formează un fel de înveliş (capsulă) ce înconjoară mesajul original, de aceea acest procedeu se mai numeşte şi *procedeul de încapsulare* a datelor.

Dispozitivele ce activează la nivelul 2 OSI se numesc *switch-uri (comutatoare)* şi *bridge-uri (punți)*. Dacă ruterele ce activează la nivelul rețea iau decizii privind direcționarea pachetelor de date inter-rețele, switch-urile şi bridge-urile se ocupă cu identificarea destinației pachetelor în cadrul rețelei locale. Atunci când un sistem terminal sursă doreşte să transmită date către un sistem terminal destinație, nivelul legătură de date foloseşte adresarea hardware (spre deosebire de nivelul rețea care foloseşte

adresarea software). Atunci când un pachet de date este schimbat între două rutere, este transformat într-un cadru prin adăugarea informației de control la nivelul legătură de date, dar această informație este înlăturată de către ruterul destinație şi este considerat doar pachetul original de date. Acest procedeu de încapsulare în cadre a pachetului continuă până ce pachetul ajunge la destinație. Este important să înțelegem că pachetul de date rămâne intact pe toată durata transportului, încapsularea cu diverse tipuri de informație de control (în funcție de mediul de transmisie) neafectând conținutul său.

Nivelul legătură de date are două subnivele:

***Subnivelul MAC (Media Access Control) 802.3*** care defineşte modalitatea în care pachetele sunt trimise spre *mediul de transmisie*. Accesul la mediul de transmisie (pe scurt, *mediu*) se face pe baza regulii *primul venit, primul servit,* atunci când este partajată întreaga lățime de bandă. Tot aici se defineşte şi adresarea fizică precum şi topologia locală. De asemenea, la acest subnivel pot apărea funcții legate de *notificarea erorilor*, *transmiterea în ordine a cadrelor* sau *controlul fluxului*.

***Nivelul LLC (Logical Link Control) 802.2*** se ocupă cu identificarea protocoalelor de nivel superior pe care le încapsulează. Un header LLC specifică nivelului legătură de date cum să acționeze cu un pachet de date atunci când se primeşte un cadru. De exemplu, un sistem terminal ce recepționează un cadru examinează header-ul LLC pentru a vedea dacă pachetul este detinat protocolului IP de la nivelul rețea. Subnivelul LLC poate oferi, de asemenea, controlul fluxului şi controlul secvențelor de biți.

### *5.4.3.7 Nivelul fizic*

Nivelul fizic din modelul OSI se ocupă cu recepția şi transmiterea şirurilor de biți, comunicând direct cu diversele tipuri de medii de transmisiune. Fiecare mediu de transmisie reprezintă valorile biților (0 sau 1) în mod diferit. Astfel, unele dintre acestea folosesc pentru reprezentare tonuri audio, altele tranziții de stare (schimbări de voltaj) etc. Pentru fiecare mediu de transmisiune în parte este necesar să se specifice un protocol pentru specificarea modelelor de biți folosite, modalitatea de codificare a biților în semnale media precum şi diverse caracteristici fizice ale interfețelor şi conectorilor corespunzătoare mediilor de transmisiune.

Specificaţiile nivelului fizic includ caracteristicile electrice, mecanice, procedurale şi funcţionale necesare pentru activarea, administrarea şi dezactivarea legăturilor fizice între sistemele terminale din reţea. La nivelul fizic sunt identificate, spre exemplu, interfeţele între echipamentele terminale DTE (Data Terminal Equipment) şi echipamentele DCE (Data Circuit-Terminating Equipment). Echipamentele DCE sunt de regulă localizate la furnizorul de servicii de reţea în timp ce echipamentele DTE sunt echipamentele ataşate. Serviciile disponibile unui DTE sunt cel mai des accesate prin intermediul unui modem sau a unui echipament CSU/DSU (Channel Service Unit / Data Service Unit). Conectorii existenţi la nivelul fizic precum şi diferitele topologii fizice sunt definite de către OSI ca şi *standarde*, permiţând comunicaţia dintre sisteme terminale diferite.

Dispozitivele de reţea ce acţionează la nivelul fizic sunt repetoarele şi hub-urile. Un *repetor (repeater)* recepţionează un semnal digital pe care îl amplifică şi îl trimite mai departe sau îl regenerează, direcţionându-l către toate porturile active de ieşire fără a examina conţinutul datelor. Hub-ul acţionează în acelaşi mod, fiind de fapt un repetor multiport. Fiecare semnal digital recepţionat la un port de intrare al hub-ului este regenerat sau amplificat şi trimis apoi către toate porturile hub-ului. În acest caz, toate dispozitivele ataşate hub-ului spunem că se află în acelaşi *domeniu de coliziuni* şi în acelaşi *domeniu broadcast*. Un domeniu broadcast este un domeniu în care echipamentele din segmentul de reţea respectiv "ascultă" toate semnalele de tip broadcast de pe acel segment. La fel ca şi repetoarele, hub-urile nu analizează traficul de date pe timpul retransmiterii lor. Hub-urile generează o reţea fizică de tip *stea (star)* în care hub-ul este nodul central la care sunt conectate dispozitivele în formă de stea. Standardul *Ethernet* pentru reţele LAN foloseşte o topologie logică de tip magistrală (bus), în care semnalele sunt transmise de la un capăt la altul al reţelei şi fiecare echipament conectat la hub "ascultă" dacă un alt echipament transmite date.

### 5.4.4 Modelul TCP/IP

#### *5.4.4.1 Introducere*

Modelul de referinţă TCP/IP este modelul utilizat de strămoşul tuturor reţelelor de calculatoare, care este ARPANET-ul şi de către succesorul acestuia, Internetul. Concepută iniţial ca o reţea extrem de fiabilă

şi care să ofere posibilitatea interconectării diferitor tipuri de echipamente, rețeaua ARPANET a Departamentului de Apărare a SUA a condus în cele din urmă la realizarea arhitecturii TCP/IP sau, altfel spus, *modelul de referință TCP/IP*. Arhitectura TCP/IP a fost definită pentru prima dată în 1974 de către autorii *Cerf şi Kahn*. Ulterior ea este dezvoltată de către Leiner în 1985 iar filozofia din spatele modelului este analizată de către Clark (1988). Modelul TCP/IP conține 4 nivele:

- Nivelul gazdă-la-rețea *(host-to-network)*
- Nivelul Internet
- Nivelul transport
- Nivelul aplicație

Figura 6.12 ne prezintă comparativ modelul TCP/IP şi modelul OSI.

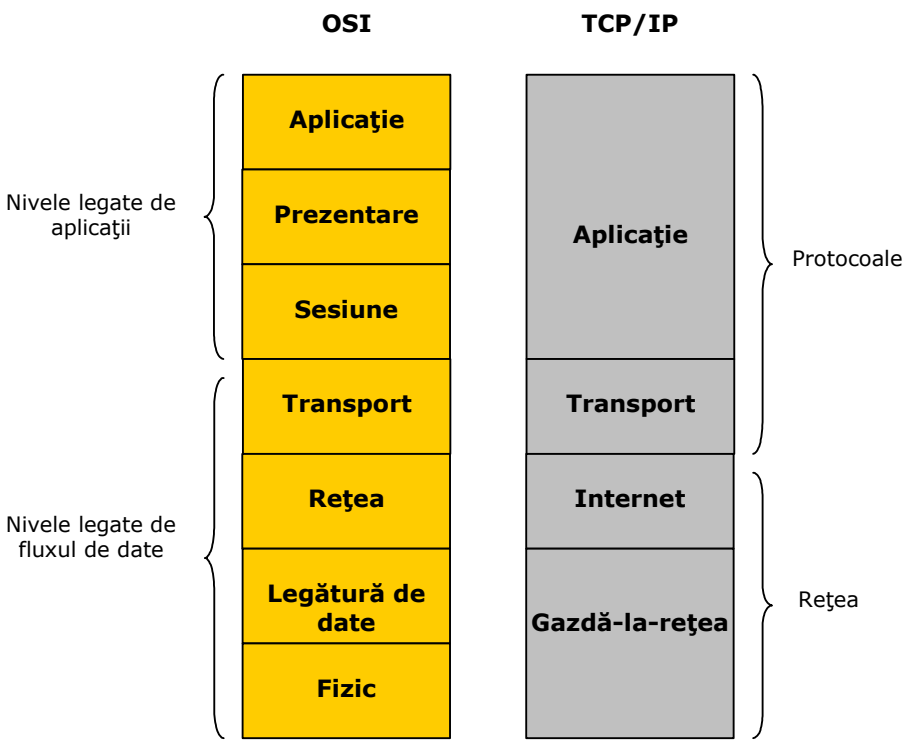



Fig. 5.12 Comparație între modelul TCP/IP şi modelul OSI

#### *5.4.4.2 Nivelul aplicație*

Spre deosebire de modelul OSI, modelul TCP/IP nu conține nivelurile sesiune şi prezentare, ele nefiind incluse deoarece nu s-a simțit nevoia lor. Experiența oferită de modelul OSI a dovedit că această viziune a fost corectă deoarece în majoritatea aplicațiilor cele două nivele nu aduc un real folos. Practic, funcțiunile nivelelor sesiune şi prezentare din modelul OSI au fost în cazul modelului TCP/IP încorporate în nivelul aplicație.

Nivelul aplicație se situează deasupra nivelului transport şi are în componență toate protocoalele de nivel înalt. Printre aceste protocoale se află TELNET (pentru conexiuni pe calculatoare la distanță), FTP (File Transfer Protocol - transfer de fişiere), SMTP (Simple Mail Transmission Protocol – pentru poşta electronică), DNS (Domain Name Service – pentru stabilirea corespondenței între numele gazdelor şi adresa de rețea) şi HTTP (HyperText Transfer Protocol – pentru aducerea paginilor web).

#### *5.4.4.3 Nivelul transport*

Este următorul nivel de sus în jos după nivelul aplicație şi este astfel conceput încât să permită conversații între entitățile pereche din sistemul terminal sursă şi cel destinație, într-un mod asemănător cu nivelul transport din modelul OSI. La acest nivel se află implementate două protocoale: TCP (Transmission Control Protocol) şi UDP (User Datagram Protocol). TCP este protocolul sigur şi fiabil orientat pe conexiune care permite ca un flux de octeți trimişi de la o sursă să ajungă la destinație fără erori, în timp ce UDP este un protocol nesigur, fără conexiune, destinat aplicațiilor care doresc să utilizeze propria lor secvențiere şi control al fluxului şi nu cele oferite de TCP.

#### *5.4.4.4 Nivelul Internet*

Nivelul Internet reprezintă coloana vertebrală a întregii arhitecturi TCP/IP. Rolul acestui nivel este acela de a permite sistemelor gazdă să trimită pachete în orice rețea şi de a face ca pachetele să circule independent până la destinație. Pachetele de date pot sosi într-o ordine diferită de aceea în care au fost transmise, caz în care rearanjarea lor în ordinea corectă trece

în sarcina nivelelor superioare. La nivelul Internet se definește un format standard de pachet de date (pachet IP) și un protocol denumit IP (Internet Protocol), sarcina nivelului fiind aceea de a livra pachetele IP către destinație. Probleme apar la dirijarea pachetelor în rețea și la evitarea congestiei; de aceea putem spune că nivelul Internet din modelul TCP/IP acționează ca și nivelul rețea din modelul OSI.

Se poate face o analogie a funcționalității acestui nivel cu aceea a sistemului poștal internațional. Atunci când trimitem o scrisoare la destinație sistemul poștal știe să direcționeze scrisoarea către destinatar, dar noi nu cunoaștem mecanismele prin care se face acest lucru; singurul lucru care ne interesează este ca scrisoarea să ajungă în bune condițiuni și în timp util la destinație.

#### *5.4.4.5 Nivelul gazdă-la-rețea*

Nivelul gazdă-la-rețea (denumit și *nivelul de acces la rețea*) este cel mai puțin specificat dintre toate nivelele modelului TCP/IP. Numele său este oarecum prea general și confuz; acest nivel se ocupă cu toate chestiunile legate de conexiunile *fizice* pe care trebuie să le străbată pachetele IP pentru a ajunge în bune condițiuni la destinație. El include specificații tehnologice legate de rețele LAN și WAN, precum și toate detaliile corespunzătoare nivelelor fizic și legătură de date prezente în modelul OSI.

### 5.5 Internetul și rețelele de calculatoare

#### *5.5.1 Introducere și concepte de bază*

Internetul este un termen nou, intrat de curând în vocabularul nostru de zi cu zi. Unii chiar abordează subiecte legate de "sindromul Internet" sau dependența de Internet. Vom încerca în cele ce urmează să clarificăm definiția acestuia, având în vedere că până și mulți utilizatori împătimiți ai Internetului nu știu să facă deosebirea între Internet și Web sau între serviciul de poștă electronică și cel de transfer de fișiere.

Trebuie să lămurim de la început că nu se poate da o definiție complexă a termenului de Internet în câteva rânduri. Având însă câteva

noţiuni de bază şi o serie de caracteristici lămurite, ne putem face o privire de ansamblu asupra concepţiei de Internet.

În primul rând, Internetul este o reţea de calculatoare (este, de fapt, o *reţea de reţele*) la nivel mondial prin intermediul cărora sunt interconectate milioane de echipamente de calcul (aici sunt incluse şi calculatoarele personale) din întreaga lume. O reţea de calculatoare reprezintă o *colecţie de calculatoare interconectate* între ele, adică sunt capabile să schimbe informaţie între ele. Pe de altă parte, Internetul este denumirea celei mai vaste grupări de surse de informaţie din lume. Reţeaua de care vorbeam mai înainte are o dimensiune extinsă la mărimea planetei noastre şi cuprinde o cantitate inimaginabilă de resurse fizice, logice, informaţionale.

Printre echipamentele interconectate se găsesc: calculatoare personale, staţii de lucru Unix, servere de Web sau de e-mail, laptop-uri, pagere, telefoane mobile, etc. De curând au fost conectate la Internet şi dispozitive electrocasnice, cum ar fi frigiderul sau cuptorul cu microunde. Se prevede că în viitor multe dintre echipamentele electrocasnice vor dispune de conexiune Internet. Toate aceste echipamente sunt denumite sisteme gazdă (*hosts* sau *end systems*). Aplicaţiile Internet care ne sunt tuturor foarte familiare (poşta electronică sau web-ul) sunt de fapt, aplicaţii de reţea ce rulează pe aceste sisteme gazdă.

Pentru a comunica între ele, sistemele gazdă folosesc aşa numitele *protocoale* pentru controlul transmiterii, recepţiei şi corecţiei informaţiilor care circulă prin Internet. Mai multe despre protocoale în continuarea capitolului. Dintre aceste protocoale, TCP (Transmission Control Protocol) şi IP (Internet Protocol) sunt cele mai importante protocoale folosite în Internet. De asemenea, pentru asigurarea conexiunii între ele, sistemele gazdă folosesc legături de comunicaţie ce constau din diverse tipuri de cabluri, printre care cablu coaxial, torsadat, fibră optică sau pot fi conexiuni fără fir, prin unde radio, de exemplu. Una dintre caracteristicile importante ale acestor legături este viteza teoretică de transfer a datelor care este denumită lăţime de bandă (*bandwidth*) şi care se exprimă în biţi sau multipli ai acestora pe secundă (1 Mb/s = 1000 biţi/s).

Sistemele gazdă nu sunt interconectate direct între ele, ci prin intermediul unor dispozitive intermediare denumite rutere. Pe scurt, un ruter este un dispozitiv care preia informaţia ce ajunge la el prin intermediul uneia dintre legăturile (de intrare) de comunicaţie şi o trimite mai departe pe o altă legătură (de ieşire) de comunicaţie. Formatul informaţiilor care sunt recepţionate şi transmise mai departe între rutere şi sistemele gazdă sunt precizate de protocolul IP. Acest protocol reprezintă *"limbajul universal"* al

Internetului şi de aceea se mai numeşte şi *"Internet dial tone"*. Drumul pe care îl parcurg informațiile de la transmițător la receptor poartă numele de *rută* (*route* sau *path*) în rețea.

Modalitatea de stabilire a unei conexiuni în Internet (pentru a putea transmite informații de la un transmițător la un receptor) se bazează pe o tehnică denumită *comutare de pachete*, care permite mai multor sisteme să comunice pe o rută (sau o porțiune dintr-o rută) Internet, în acelaşi timp. Topologia Internetului (structura sistemelor conectate la Internet) este ierarhizată în modul următor: la bază sunt sistemele gazdă conectate la un ISP (Internet Service Provider - Furnizor de Servicii Internet) local prin intermediul unor rețele de acces, furnizorii locali sunt conectați la nişte furnizori naționali sau internaționali, iar aceştia din urmă sunt conectați împreună la cel mai înalt nivel din această ierarhie.

Este interesant de remarcat faptul că pot fi adăugate noi componente sau nivele (noi rețele sau noi rețele de rețele) în această topologie ierarhică într-o manieră foarte simplă, aşa cum ai adăuga noi piese într-un joc de Lego. Internetul a crescut în ultimii ani şi continuă să crească într-un ritm exponențial: dacă în prima jumătate a lui 1996 au fost adăugate aproximativ 3.000.000 de host-uri *(sisteme gazdă)* [Network 1996], în prima jumătate a lui 2001 au apărut 16.000.000 de noi host-uri conectate la Internet [Network 2001].

Această rețea uriaşă care este Internetul nu ar fi s-ar fi putut crea dacă nu ar fi fost create, testate şi implementate o serie de standarde. Dacă în faza incipientă a rețelelor de calculatoare (anii 1970) era imposibil să interconectezi computere provenite de la diverşi producători (un computer IBM cu un server Gateway, spre exemplu) datorită incompatibilității protocoalelor folosite, treptat s-a ajuns la concluzia că trebuie folosite standarde generalizate de comunicație pentru a putea interconecta diverse echipamente provenite de la diverşi producători. Astfel au luat ființă aşa numitele standarde deschise *(open-standards)* şi necesitatea conectivității indiferent de platformă *(cross-platform)*. Aceste standarde sunt dezvoltate de organisme internaționale specializate, precum IETF (Internet Engineering Task Force) ale căror documente poartă denumirea de documente RFC (Request For Comments). După cum reiese şi din denumirea originală (cereri pentru observații, comentarii), RFC-urile s-au născut pentru a rezolva problemele arhitecturale ale predecesorilor Internetului.

Documentele RFC au evoluat astfel încât acum sunt adevărate standarde, documente tehnice şi detaliate, care cuprind definiții de

protocoale cum ar fi TCP (Transmission Control Protocol), IP (Internet Protocol), SMTP (Simple Mail Transfer Protocol) sau HTTP (Hyper Text Transfer Protocol). La ora actuală există peste 2000 de documente RFC. Copii ale documentelor RFC sunt găzduite de numeroase site-uri Web. Una dintre aceste locaţii este la Institutul de Ştiinţe ale Informaţiei (ISI – Information Sciences Institute - http://www.rfc-editor.org). Alte locaţii sunt: http://www.faqs.org/rfcs, http://www.pasteur.fr/other/ computer/RFC şi http://www.csl.sony.co.jp/rfc/.

Internetul este un domeniu *public*, care cuprinde, după cum am mai spus, o varietate de reţele publice ale unor companii private, instituţii educaţionale sau guvernamentale. Există însă şi reţele asemănătoare *private*, ale căror calculatoare gazdă nu sunt accesibile din afara reţelei respective. O astfel de reţea se numeşte *intranet* şi de regulă foloseşte aceleaşi tehnologii ca cele folosite în Internet.

### *5.5.2 Internet – scurt istoric*

Funcţionarea Internetului se bazează pe protocolul TCP/IP, care este de fapt o colecţie de protocoale dintre care cele mai importante sunt TCP şi IP, de unde provine şi denumirea de TCP/IP. De multe ori întâlnim denumirea de stivă de protocoale TCP/IP (TCP/IP Protocol Stack). Pentru a înţelege funcţionarea unei reţele TCP/IP (bazată pe TCP/IP) trebuie să cunoaştem o serie de considerente care au stat la baza naşterii reţelei Internet şi a stivei de protocoale TCP/IP. Dacă la începuturile dezvoltării sistemelor de calcul comerciale (sfârşitul anilor 1960, începutul anilor 1970) companiile foloseau arhitecturi şi protocoale proprietare (ce aparţineau unui anumit producător), treptat s-a ajuns la nevoia de a achiziţiona echipamente provenite de la mai mulţi producători. Aceste echipamente trebuiau interconectate şi făcute să lucreze împreună pe baza aceloraşi standarde şi protocoale.

Utilizarea computerelor în programul agenţiei americane de stat ***ARPA*** *(Advanced Research Projects Agency)* din cadrul Departamentului de Apărare *(DOD – Department Of Defense)* a condus la elaborarea coordonării dezvoltării unei reţele independente de producător pentru a conecta marile centre de cercetare. Necesitatea unei astfel de reţele a fost prima prioritate a acestui program, având în vedere că, iniţial, fiecare centru de cercetare folosea tehnologii proprietare. Anul 1968 a marcat începutul

elaborării unei rețele bazate pe *comutare de pachete*, care mai târziu a devenit rețeaua **ARPAnet**.

Rețeaua ARPAnet a fost prima rețea de calculatoare de arie largă (WAN – Wide Area Network) din lume, concepută să permită unor pachete de date să fie rutate în rețea ca entități de sine stătătoare. Acest lucru a reprezentat o revoluție în domeniu, deoarece rețelele anterioare se bazau pe *comutarea de circuite*, ce însemna stabilirea de conexiuni dedicate între două locații pentru ca acestea să poată schimba date. Mai mult, ARPAnet oferea pentru prima dată posibilitatea interconectării mai multor locații într-o topologie neregulată, permițând datelor să circule între oricarea dintre aceste locații pe diverse rute. Conceptul de bază era următorul: dacă una din locații era distrusă (eventual bombardată într-un război – să nu uităm că era vorba despre un proiect militar), acest lucru nu afecta comunicațiile dintre celelalte locații care făceau parte din rețea.

În aceeaşi perioadă de timp şi alți furnizori de servicii de rețea au început să dezvolte conexiuni cu locații din rețeaua ARPAnet, ceea ce a dus treptat la apariția termenului de **Internet**. În anii următori tot mai multe organizații au fost adăugate în ARPAnet, în paralel cu dezvoltarea altor rețele şi tehnologii de rețea, precum Ethernet.

Toate aceste dezvoltări ulterioare au condus la concluzia că este nevoie de o serie de protocoale de rețea care să opereze la un nivel superior celui fizic, astfel încât să se permită schimbul de informații între diverse rețele fizice. Aceste protocoale trebuiau implementate în software deasupra oricărei topologii de rețea, indiferent că era vorba despre o rețea de arie largă WAN cu comutare de pachete (ARPAnet) sau o rețea locală LAN (Local Area Network) Ethernet sau Token Ring.

### *5.5.3 Definiția unui protocol de rețea*

Unul din termenii cei mai folosiți atunci când este vorba de o rețea de calculatoare sau de Internet este termenul de "protocol". Vom prezenta în continuare o definiție şi câteva exemple pentru a putea identifica un protocol. Probabil că cea mai bună modalitate de a înțelege noțiunea de protocol este aceea de a considera pentru început o serie de analogii cu intercomunicarea din lumea umană. Să considerăm exemplul în care întrebăm pe cineva unde se află o anume stradă (figura 5.13).

Bunele maniere (protocolul uman) ne fac să spunem întâi "Bună ziua!" pentru a începe comunicarea cu o altă persoană. Răspunsul ar trebui să fie, desigur, tot "Bună ziua!", ca o confirmare a faptului că este acceptată comunicarea. Interpretarea răspunsului ca un accept al comunicării ne permite acum să formulăm întrebarea care ne interesează. Dacă raspunsul inițial al persoanei căreia îi adresăm "Bună ziua!" ar fi fost "Lasă-mă în pace, sunt ocupat!" sau ceva asemănător, atunci ar fi însemnat că nu există posibilitatea comunicării. În acest caz, nu mai are rost să formulăm întrebarea al cărei răspuns dorim să-l aflăm. Uneori este posibil să nu primim nici un răspuns la o întrebare, caz în care de regulă renunțăm a mai repeta întrebarea.

Regulile intercomunicării umane (protocolul uman) sunt astfel reprezentate de mesajele pe care le trimitem şi de acțiunile specifice pe care le întreprindem corespunzătoare răspunsului primit de la interlocutor sau producerii altor evenimente. Mesajele transmise şi cele recepționate joacă un rol fundamental în cazul protocoalelor umane; dacă o persoană are obiceiuri diferite sau foloseşte un limbaj străin altei persoane, atunci protocoalele diferite nu vor permite intercomunicarea între respectivele persoane. Acelaşi lucru este valabil şi în cazul comunicării între entitățile dintr-o rețea de calculatoare. Pentru a putea comunica, respectivele entități trebuie să folosească (să ruleze) acelaşi protocol de rețea.

Un protocol de rețea este asemănător unui protocol uman, excepție făcând obiectele comunicării: în loc să avem de-a face cu oameni, avem de-a face cu componente hardware sau software ale rețelei. Toate activitățile dintr-o rețea de calculatoare (deci şi din Internet) sunt bazate pe funcționarea unui anumit set de protocoale. De exemplu, comunicarea dintre două calculatoare în rețea se face prin protocoale implementate în hardware la nivelul plăcii de rețea pentru controlul fluxurilor de biți transmişi prin intermediul suportului fizic; protocoalele de control al congestiilor au grijă să controleze viteza de transmitere a datelor între un transmițător şi un receptor iar protocoalele de poştă electronică guvernează modalitatea de transmitere şi de recepție a mesajelor de tip e-mail.

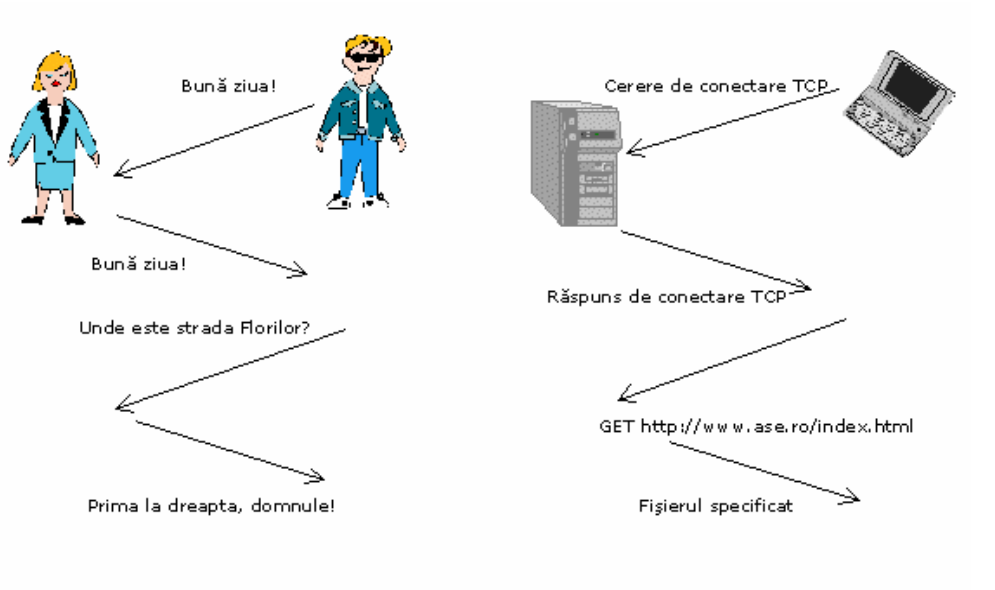



Fig. 5.13 Analogie între un protocol uman şi un protocol de rețea

În figura 5.13 este prezentat cazul în care un calculator face o cerere unui server Web (asta se întâmplă în momentul în care scriem adresa web în fereastra browserului), se primeşte un răspuns afirmativ de conexiune din partea serverului şi apoi calculatorul foloseşte un mesaj de tip "GET" pentru a recepționa pagina respectivă. În cele din urmă, serverul returnează conținutul fişierului calculatorului care a făcut cererea.

Ca urmare a analogiei cu comportamentul uman, putem da următoarea definiție a protocolului: *un protocol defineşte formatul şi ordinea mesajelor schimbate între două sau mai multe entități ce comunică între ele, precum şi acțiunile ce sunt întreprinse odată cu transmiterea sau recepția unui mesaj sau a unui alt eveniment*.

### *5.5.4 Naşterea stivei de protocoale TCP/IP*

Naşterea Internetului a dus în 1973 la începutul dezvoltării stivei de protocoale TCP/IP, care se dorea a fi o colecție de protocoale de rețea bazate pe software care să permită *oricărui* sistem să se conecteze cu *orice* alt sistem, folosind *orice topologie* de rețea. Cinci ani mai târziu, în 1978, era gata versiunea 4 IP, adică aceeaşi versiune pe care o folosim şi astăzi.

Imediat după aceea au început să apară o serie de semnale pozitive în legătură cu recunoaşterea TCP/IP: Universitatea Berkeley din California a încorporat stiva de protocoale TCP/IP în versiunea proprie de UNIX-FreeBSD (distribuită gratis) ce avea să devină cel mai folosit sistem de operare în comunităţile academice şi de cercetare.

Introducerea la scară largă a suitei de protocoale TCP/IP a produs o serie de schimbări majore în lumea reţelelor de calculatoare. În primul rând, topologia de bază a unei reţele era concentrată pe un nod central, în care fiecare sistem ataşat trimitea datele unui nod central (pe post de dispecer) pentru a fi procesate. Cu alte cuvinte, utilizatorii din reţea nu aveau independenţă în lucru, orice procesare, tipărire la imprimantă etc. trebuind să treacă pe la nodul central.

O dată cu introducerea TCP/IP, lucrurile s-au schimbat: s-a introdus "descentralizarea", astfel încât fiecare echipament din reţea era tratat independent şi complet funcţional, fără a mai depinde de un nod central. Comunicarea cu alt echipament din reţea se putea face acum direct, fără să se comunice mai întâi cu nodul central. Reţelele bazate pe protocolul IP sunt oarecum anarhice, fiecare echipament acţionând pe cont propriu ca o unitate autonomă, responsabilă pentru serviciile de reţea proprii [Hall 2000]. Această concepţie arhitecturală a permis partajarea aplicaţiilor şi a resurselor la scară largă, având în vedere că un model centralizat *top-down* nu era viabil în cazul existenţei a milioane de echipamente larg răspândite. În plus, acest model oferea siguranţă în exploatare în cazul "căderii" unei componente din reţea, în contrast cu modelul centralizat în care toată funcţionarea se oprea în cazul "căderii" nodului central.

### *5.5.5 Arhitectura Internetului*

De-a lungul timpului, ARPAnet a evoluat într-o "reţea de reţele", folosind TCP/IP şi conectând între ele diverse organizaţii comerciale, educaţionale, politice etc. A rezultat astfel o structură generală destul de neregulată a Internetului (figura 5.14).

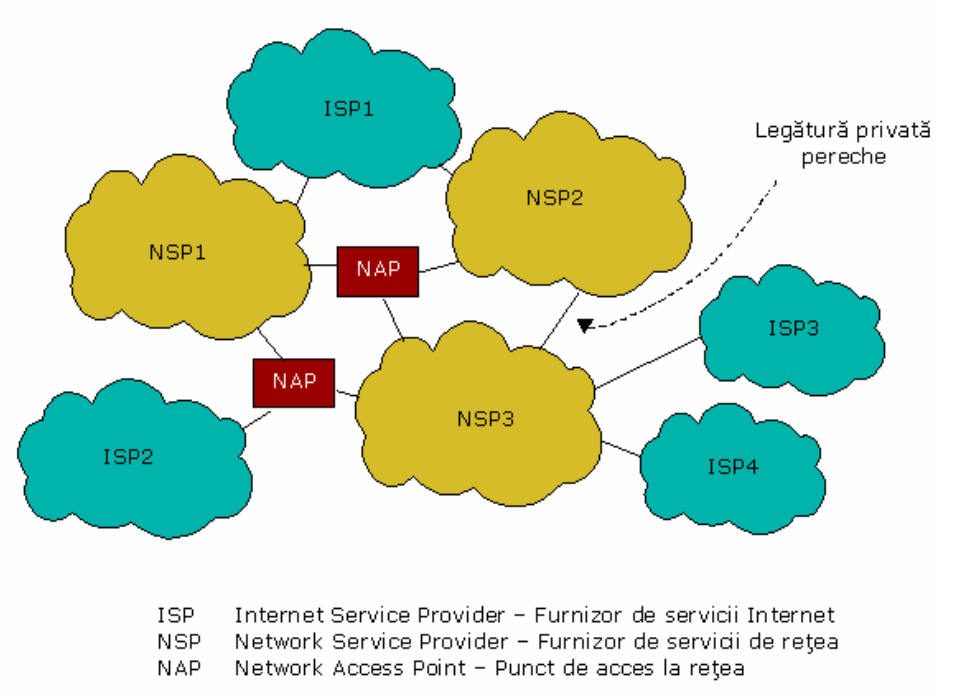



Fig. 5.14 Arhitectura generală a Internetului

În rețeaua Internet există:

- Furnizorii de servicii Internet care asigură clienților acces la Internet - **ISP** (*Internet Service Provider*);
- Furnizorii de servicii de rețea, care asigură conexiunile între furnizorii de acces la Internet din întreaga lume - **NSP** (*Network Service Provider* sau *Backbone Provider*);

În figură se mai pot observa așa numitele **NAP** (*Network Access Points*) care reprezintă punctele de acces la rețea.

În zilele noastre se tinde spre o arhitectură mai structurată, organizată ierarhic ca un arbore (figura 5.15). La primul nivel din arbore se află câțiva furnizori de servicii care oferă majoritatea serviciilor de interconectare la nivel unei țări sau la nivel mondial, care se numesc furnizori naționali. Majoritatea acestor firme sunt mari companii de telecomunicații specializate în rețele de scară largă.

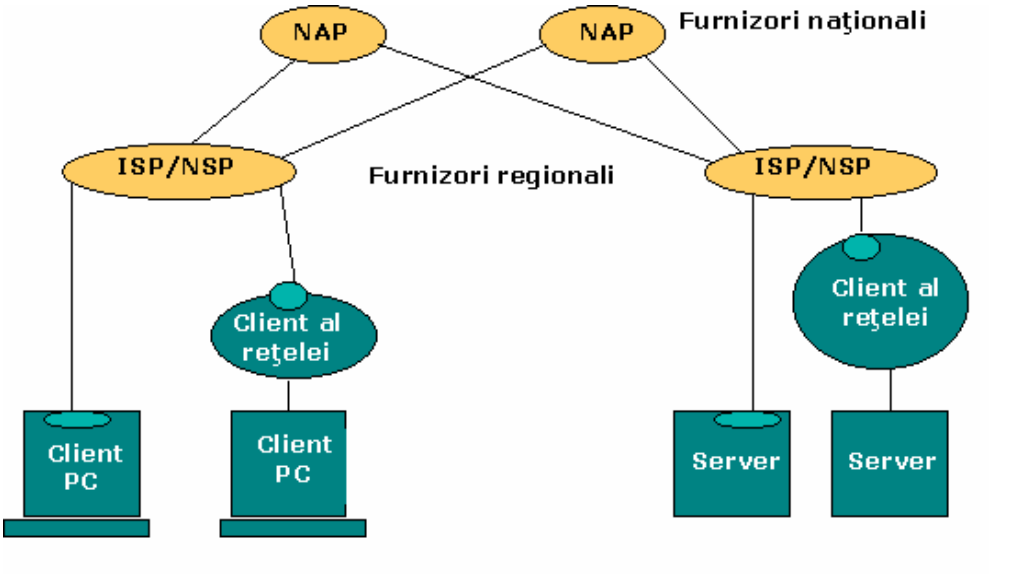



Fig. 5.15 Structura ierarhică a Internetului

Pe următorul nivel în arbore găsim furnizorii de acces Internet sau de servicii Internet (întâlniți uneori şi sub denumirea de **IAPs** – *Internet Access Providers*) pe care îi denumim furnizori regionali. Aceştia oferă acces la nivel unei localități sau regiuni geografice restrânse, având servicii de conexiune la viteze mai reduse decât furnizorii naționali.

Următorul nivel îl constituie clienții rețelei (fie ei clienți PC sau servere) ce sunt conectați prin intermediul unui furnizor de servicii Internet la un punct de acces la rețea, de unde se face legătura spre întreaga rețea Internet. Punctele de acces la rețea sunt, de regulă, bazate pe tehnologie ATM şi FDDI.

### *5.5.6 Componente Internet*

#### *5.5.6.1 Calculatoare gazdă, clienți şi servere*

Computerele folosite în jargonul rețelelor de calculatoare sunt denumite, de regulă, calculatoare gazdă (hosts) sau sisteme terminale (end-systems). Denumirea de calculator gazdă provine de la faptul că acesta găzduieşte programe de nivel aplicație (program de e-mail, navigator web sau program de chat). Denumirea de sisteme terminale provine de la faptul că ele se află la "marginea" Internetului (vezi figura 5.16).

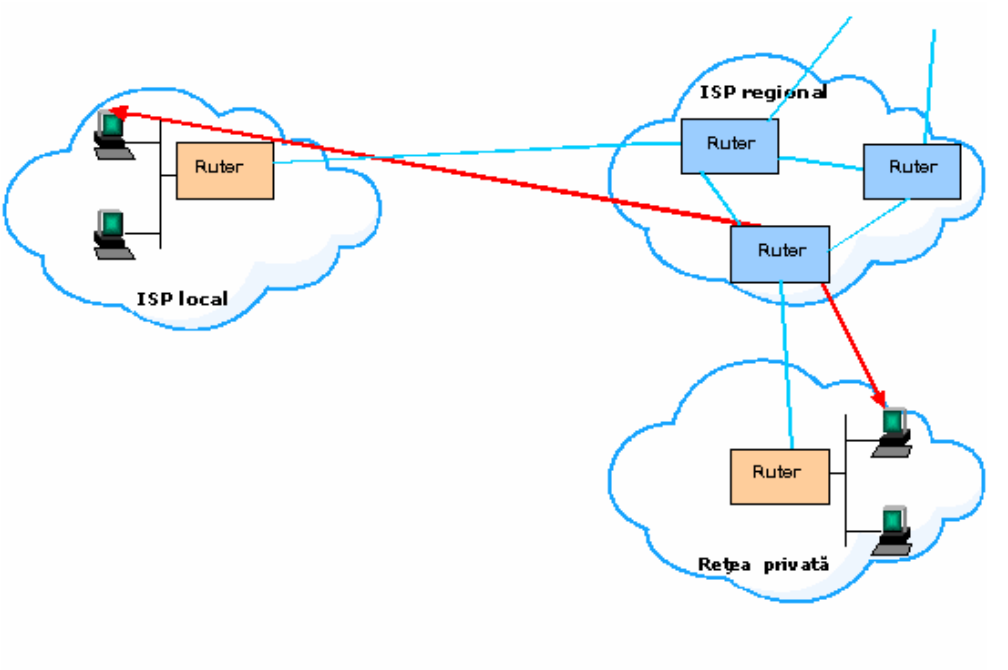



Fig. 5.16 Componente ale Internetului

Calculatoarele gazdă se împart în două categorii: clienți şi servere. Clienții sunt reprezentați de calculatoare PC sau stații de lucru, în timp ce serverele sunt calculatoare mai puternice care au de obicei funcționalități specifice: pot fi servere de baze de date, servere de mail sau servere de Web, etc. Sintagma client/server este însă cu mult mai importantă în cadrul rețelelor de calculatoare, având în vedere faptul că majoritatea acestora folosesc modelul client/server. Conform acestui model, un program client ce rulează pe un sistem terminal cere şi primeşte informații de la un program server ce rulează pe alt sistem terminal. Cele mai multe aplicații Internet (Web, e-mail, ftp, telnet) folosesc acest model; datorită faptului că un program client rulează pe un calculator şi programul server rulează alt calculator, aceste aplicații se mai numesc şi aplicații distribuite.

Dacă cele mai multe sisteme terminale sunt formate din calculatoare personale, stații de lucru şi servere, în ultimul timp au apărut din ce în ce mai multe echipamente conectate la Internet ca sisteme terminale (camere digitale, sisteme WebTV etc.) [Dertouzous 1999].

### *5.5.6.2 Servicii orientate pe conexiune*

Sisteme terminale aflate în reţea comunică între ele şi fac schimb de informaţii conform unui protocol de comunicaţie. Serverele, ruterele, legăturile fizice şi alte componente ale Internetului oferă mijloacele transportării acestor mesaje între aplicaţiile sistemelor terminale. Serviciile de conexiune oferite se împart în două tipuri: servicii neorientate pe conexiune *(connectionless)* şi servicii orientate pe conexiune *(connection-oriented)*.

În cazul serviciilor bazate (orientate) pe conexiune, programele client şi server trimit pachete de control unul altuia înainte de a trimite pachetele cu date. Acest procedeu se numeşte *handshaking* ("strângere de mână") şi are rolul de a atenţiona atât clientul cât şi serverul că urmează să aibă loc schimbul de date.

O dată încheiată procedura de *handshaking*, se spune că se stabileşte o conexiune între cele două sisteme terminale. Sistemele terminale sunt informate despre această conexiune dar modalitatea prin care pachetele care sunt trimise de la sursă la destinaţie prin Internet nu permite stocarea unor informaţii legate de starea conexiunii. Serviciul orientat pe conexiune din Internet oferă şi alte servicii ce ţin de transferul sigur al datelor, de controlul fluxului sau de controlul congestiilor.

Transferul *sigur* al datelor *(reliable data transfer)* înseamnă faptul că o aplicaţie se poate baza pe conexiune pentru a transmite datele fără erori şi în ordinea corectă. Siguranţa transmisiei datelor în Internet se face prin utilizarea confirmărilor şi a retransmisiilor. Pentru a ne face o idee generală asupra modului în care funcţionează acest lucru, să presupunem că a fost stabilită o conexiune între două sisteme terminale X şi Y. Atunci când X primeşte un pachet de la Y, îi trimite o confirmare; atunci când Y primeşte confirmarea, atunci el ştie că respectivul pachet a ajuns la destinaţie. Dacă sistemul Y nu primeşte confirmarea, atunci presupune căpachetul respectiv nu a fost primit de către X şi în acest caz retransmite pachetul.

*Controlul fluxului* este folosit pentru a ne asigura de faptul că nici una dintre părţile implicate în conexiune nu-şi aglomerează "interlocutorul" trimiţând mai multe date decât acesta este capabil să recepţioneze în unitatea de timp. Într-adevăr, o aplicaţie de la unul dintre capetele conexiunii poate să nu fie în stare să proceseze informaţia la fel de repede aşa cum ea soseşte, existând riscul supraîncărcării. Serviciul de control al fluxului forţează sistemul ce transmite datele să reducă viteza de transmitere ori de câte ori apare riscul supraîncărcării.

Serviciul de *control al congestiilor* previne intrarea într-o situaţie de blocaj. Când un ruter devine congestionat, dimensiunea memoriilor sale

tampon poate fi depăşită şi să se producă pierderi de pachete. În astfel de situații, dacă fiecare dintre sistemele pereche ce comunică continuă să trimită pachete în rețea, are loc un blocaj şi astfel puține dintre aceste pachete vor mai ajunge la destinație. Internetul evită această problemă forțând sistemele terminale să-şi reducă ratele de transfer la cer trimit pachete în rețea în astfel de perioade de congestie. Sistemele terminale sunt atenționate de existența congestiilor atunci când nu mai primesc confirmări pentru pachetele pe care le-au trimis la destinație.

Transferul sigur de date, controlul fluxului şi controlul congestiilor nu reprezintă caracteristicile generale ale unui serviciu orientat pe conexiune; orice serviciu sau protocol orientat pe conexiune are la bază doar procedura de *handshaking* pentru inițierea transferului de date între cele două sisteme terminale. Serviciul orientat pe conexiune al Internetul este **TCP** (Transmission Control Protocol), definit inițial în documentul RFC 793. Caracteristicile TCP includ transferul sigur de date, controlul fluxului şi controlul congestiilor.

### *5.5.6.3 Servicii neorientate pe conexiune*

După cum ne putem aştepta, într-un astfel de serviciu de conexiune nu există procedura de *handshaking*. În momentul în care una dintre aplicațiile ce comunică trimite pachete celeilalte aplicații, transmițătoarea trimite pur şi simplu pachetele de date. Având în vedere că nu mai există procedura inițială de stabilire a conexiunii, înseamnă că datele pot fi transmise mai rapid. Cum nu există nici confirmări de primire a pachetelor, înseamnă că cel care trimite datele nu este niciodată sigur că acestea au ajuns la destinație. Acest tip de serviciu nu se ocupă nici cu controlul fluxului, nici cu controlul congestiilor. Serviciul neorientat pe conexiune din Internet poartă numele de **UDP** (User Datagram Protocol), definit în documentul RFC 768.

Printre aplicațiile Internet ce folosesc serviciul TCP se numără: *TELNET* – conectare la distanță, *SMTP* – poştă electronică, *FTP* – transfer de fişiere, *HTTP* – World Wide Web. Exemple de aplicații Internet care folosesc UDP sunt: *Internet phone*, *audio-la-cerere* (audio-on-demand) şi *video conferința*.

# BIBLIOGRAFIE


| | | |
|---|---|---|
| 1 | [Aranghelovici 1999] | Aranghelovici Renato - *Windows 2000*, articol PC Magazine, august 1999 |
| 2 | [Bach 1986] | Bach, M.J. - *The Design of the UNIX Operating System*, Prentice Hall, Englewood Cliffs, N.J., 1986 |
| 3 | [Brăescu 1995] | Brăescu C.L., Dudaş Liviu - *Internet pentru începători*, TIMS, Bucureşti, 1995 |
| 4 | [Busschbach 1999] | Busschbach Peter - *The Internet Protocol and other things of interest, presentation at Broadband Networking Symposium*, Barcelona, 1999 |
| 5 | [Davis 2001] | Davis W., Rajkumar T.M. - *Operating Systems*, *A Systematic View - Fifth Edition*, Editura Addison Wesley, 2001 |
| 6 | [Dertouzous 1999] | M. Dertouzous, *The future of computing*, Scientific American, august 1999, pag. 52-55 |
| 7 | [Dodescu 1994] | Dodescu Gh., Vasilescu A., *Sisteme de operare MS-DOS şi UNIX*, Editura Viaţa Românească, Bucureşti, 1994 |
| 8 | [Dodescu 2002] | Dodescu Gh., Năstase F., *Sisteme de calcul şi operare*, Bucureşti, Editura Economică, 2002 |
| 9 | [Dodescu 1995] | Dodescu Gh., Năstase F., *Sisteme de calcul şi operare*, Editura Aldo, Bucureşti, 1995 |
| 10 | [Goldman 1996] | Goldman E. James - *Local Area Networks – A client/server approach* – John Wiley&Sons, Inc., 1996 |
| 11 | [Hahn 1993] | Hahn Harley, Stout Rick - *The Internet Complete Reference*, 1993 |

12 [Hall 2000] Eric A. Hall, *Internet Core Protocols: the definitive guide*, O'Reilly & Assosiates, Inc., 2000

13 [Howe 2000] Howe Walt - *A brief history of the Internet*, walthowe@delphi.com, 2000

14 [Kurose 2001] James Kurose şi Keith Ross, *Computer Networking, A Top-Down Approach Featuring the Internet* - Addison Wesley, 2001

15 [Lammle 2000] Todd Lammle, *CCNA Study Guide*, Second Edition, Sybex, 2000

16 [Leiner 1997] Leiner M. Barry, Cerf G. Vinton, Clark D. David and others, *A brief History of the Internet, Internet Society*, bleiner@computer.org, 1997

17 [Lupu 2000] Lupu Viorel - *Windows 2000 – perspectiva dezvoltatorului,* articol PC Report, februarie 2000

18 [Năstase 1999] Năstase Floarea - *Arhitectura reţelelor de calculatoare* – Editura Economică, Bucureşti 1999

19 [Network 1996] Network Wizards, *Internet Domain Survey*, July 1996, http://www.nw.com/zone/WWW-9607/report.html

20 [Network 2001] Network Wizards, *Internet Domain Survey*, July 2001, http://www.isc.org

21 [Norton 1998-1] Norton Peter - *Secrete PC* – Editura Teora, Bucureşti 1998

22 [Norton 1998-2] Norton Peter - *Ghid complet pentru Windows 95* – Editura Teora, Bucureşti 1998

23 [Norton 1999] Norton Peter - *Ghid complet pentru Windows 98* – Editura Teora, Bucureşti 1999

| | | |
|---|---|---|
| 24 | [Pitiş 1999] | Pitiş Andrei - *GNU/Linux: o alegere morală?,* articol PC Report, noiembrie 1999 |
| 25 | [Pol 2000] | Pol Daniel - *Administrarea în Windows 2000*, articol PC Report, februarie 2000 |
| 26 | [Roşca 1999] | Roşca I.G.,Ţăpuş N.,Cristea V., Atanasiu Irina, Costinescu B., Năstase Floarea, Stanciu Carmen, Paiu Octavian, Godza Gavril, *Intranet*, Editura ASE Bucureşti, 1999 |
| 27 | [Sabău 2000-1] | Sabău Mircea - *Windows 2000*, articol PC Magazine, martie 2000 |
| 28 | [Stoian 2000-1] | Stoian Adrian - *Proiectarea reţelelor Windows 2000,* articol PC Report, martie 2000 |
| 29 | [Stoian 2000-2] | Stoian Adrian - *Noi tehnologii în Windows 2000,* articol PC Report, februarie 2000 |
| 30 | [Szelag 1999] | Szelag C. Russell - *Mission possible: Delivering QoS-Capable VPNs*, Lucent Technologies, 1999 |
| 31 | [Tanenbaum 1996] | Andrew Tanenbaum, *Computer Networks, 3rd* Ed. –Prentice-Hall, 1996 |
| 32 | [Tanenbaum 1998] | *Reţele de calculatoare* – Andrew Tanenbaum, Agora, Tg. Mureş 1998 |
| 33 | [Theis 1999] | Theis T.N. - *The future of interconnection technology*, IBM Journal of Research and Development, 1999 |
| 34 | [Vasilescu 1999] | Vasilescu Adrian - *Reţele de calculatoare* - Editura Inforec, Editura ASE Bucureşti 1999 |
| 35 | [Williams 2001] | Williams R. – *Computer Systems Architecture*, A Networking Approach, Editura Addison Wesley, 2001 |
| 36 | [Zota 1999] | Zota Răzvan Daniel - *Soluţii de securitate pentru Internet*, articol Revista Informatică Economică, Editura Inforec, Bucureşti, 1999 |

37 [Zota 2002] Zota Răzvan Daniel - *Rețele de calculatoare în era Internet,* Editura Economică, Bucureşti, 2002

38 [Zota 2002] Zota Răzvan Daniel – *Sisteme de operare pentru rețele de calculatoare*, Editura Economică, Bucureşti, 2002

39 [Zota 2003] Zota Răzvan Daniel – *Sistemul de operare UNIX. Utilizare şi programare shell*, Editura ASE, Bucureşti, 2003